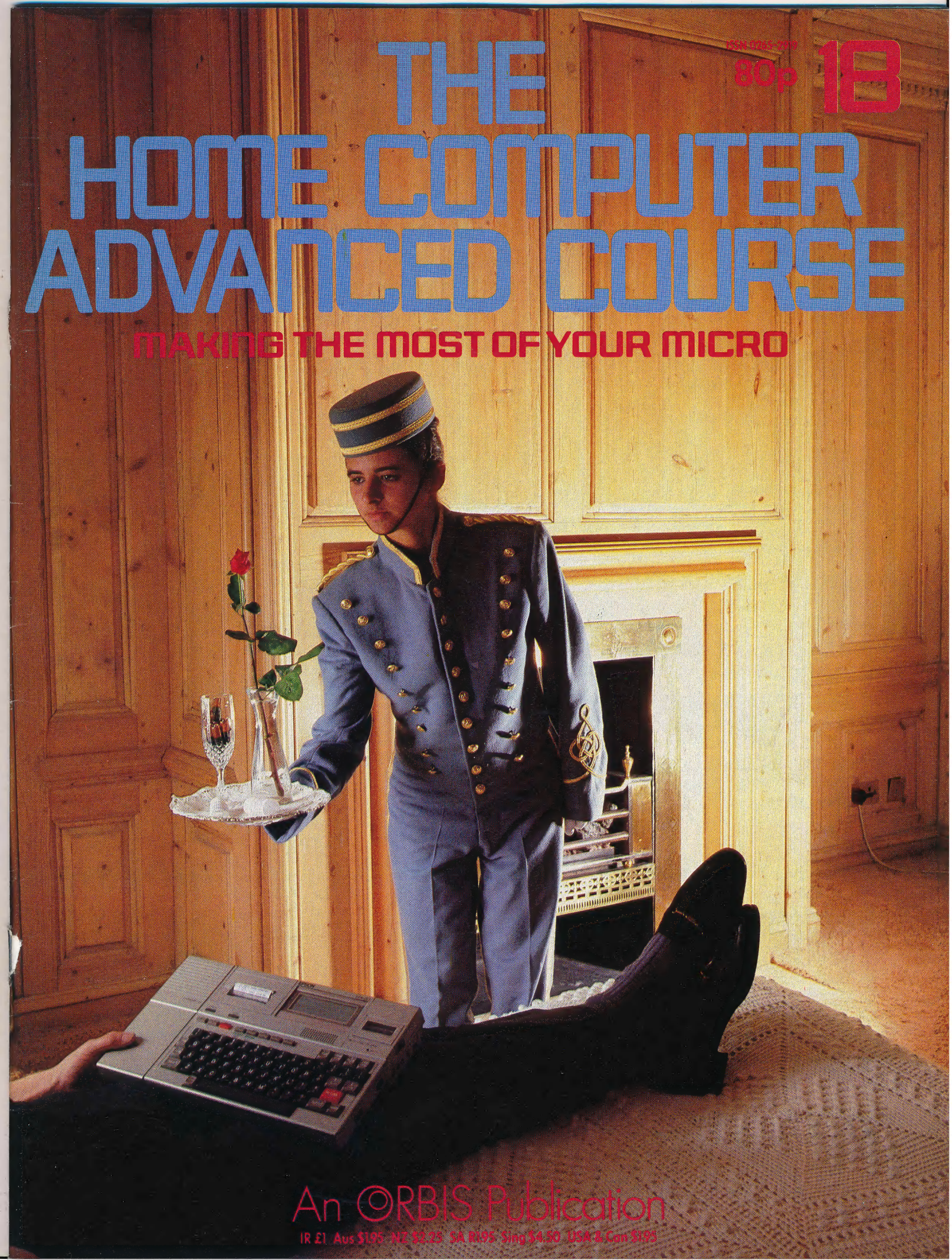
ISSN 0265-707X

80p 18

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

APPLICATION

TRAVELLING LIGHT We review the new generation of portable micros 341

HARDWARE

FIRST IMPRESSIONS Dot matrix printers can create attractive graphics 344

ADVANCED STUDIES The Advance 86a is a home micro with expansion potential 349

SOFTWARE

THE NUCLEAR FAMILY We look at a strategy game that puts your family at the controls of a nuclear war 356

JARGON

FROM CONTROL CHARACTER TO CPU A weekly glossary of terms 348

PROGRAMMING PROJECTS

JOINING FORCES We network two Spectrums to compete in a game of Battleships 346

LANDING CRAFT Our simple program will provide a intriguing game 352

PROGRAMMING TECHNIQUES

STYLE COUNSEL Continuing our series of programming tips we look at the importance of good program documentation 354

MACHINE CODE

DIRECTING THE ACTION We learn how to manipulate Spectrum sprites 357

PROFILE

RADIO CONTACT Motorola are a giant electronic components manufacturer 360

Next Week

- **The Torch disk pack turns the BBC Micro into a business computer with twin disk drives, a Z80 microprocessor (with 64K of RAM) and a full set of business software. Yet it costs little more than the Acorn disk drive.**
- **We look at how computer companies market their products and consider the strategies of some well-known manufacturers.**
- **One of the few features missing from the BBC Micro is sprites. We list a machine code program that enables a pseudo-sprite to be used from BASIC.**

QUIZ

1) Explain how the 24-bit address bus allows the Motorola 68000 to address 16 Mbytes of memory.

2) Can you think of two advantages and two disadvantages that LCD displays have against the cathode ray tube?

3) What is the function of 'attributes' on a printer?

Answers To Last Week's Quiz

A1) A print server controls the printer in a local area network.

A2) CHR$(13) results in a carriage return on a printer.

A3) The two types of video output for home computers are RGB and composite video.

A4) Machine code operations will alter the contents of the registers and so the data is PUSHed onto the stack.

QUIZ

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Contributors** Ted Ball, Max Phillips, Matt Nicholson, Geoff Nairn, Graham Storrs, Richard Pawson, Ian White; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

TRAVELLING LIGHT

When Adam Osborne launched the first portable microcomputer in 1981, it was greeted with great enthusiasm. But such has been the advance of technology that the Osborne and the many portables it inspired now appear cumbersome and heavy when compared to the latest generation of truly portable computers.

The definition of 'portable' has had to be revised since the new generation of 'hand-held' computers. Indeed, the portable micros introduced only a few years ago are now referred to as 'transportable'. True portability is now offered by computers that carry their own power supply, display, and storage devices in a package no larger than a telephone directory.

The Epson HX-20 was the first to offer this type of portability, but now its tiny 20-character by four-line liquid crystal display shows the machine's age. The latest portables such as the Tandy 100, NEC PC-8201A, and Olivetti M10 are similarly priced but can display four times as many characters on their screens.

So what can these computers do? What are their advantages and disadvantages over conventional desk-top micros? The most obvious reason to buy a portable is to have access to full computing power anywhere and at any time. Many people spend much of their time away from their desk computer, and many unproductive hours are spent in other offices, hotel rooms, airports and trains. The portable — or hand-held — computer enables this time to be put to use.

The latest generation of portables give convenient personal computing power for science and engineering work, accounting, financial management and word processing — in fact for practically any application that conventional personal computers are used for.

Hand-held computers usually carry at least three built-in programs. These are a BASIC interpreter, a word processing program, and communications software. The Tandy 100 and the Olivetti are also equipped with built-in address and scheduling programs to allow the user to find addresses, telephone numbers and daily appointments.

The communications program is extremely important as it enables the portable to communicate with other micros and databases over the telephone network. This facility can also turn the portable micro into a telex terminal or receiver and transmitter of electronic mail. Of course, a modem or acoustic coupler has to be used to achieve this. In this way, an executive away from his desk can keep in touch with his head office. A journalist on location can write his story into his portable computer and transmit it immediately to the computer back at the newspaper.

TONY SLEEP

On The Move
Computing on the move is becoming increasingly popular, mainly with businessmen. Some are using the new generation of 'hand-held' computers to snatch a few extra minutes word processing as they hop from taxi to train to plane. Others, such as salesmen, are breaking new ground by taking computers to their customers to generate on-the-spot estimates that otherwise would have taken days to prepare.

Executives on the move can relay data back to head office using a modem and ordinary telephone lines, or at the end of the day return to the office and send the data directly to a larger computer

The more expensive portable computers such as the Sharp PC-5000 and Epson PX-8 use the MS-DOS and CP/M operating systems common to their desk-top equivalents. They are therefore able to run a vast range of business software.

The Epson PX-8 comes with the popular Wordstar word processing program already installed in its ROM chips. The Sharp uses bubble memory plug-in cartridges that provide 128 Kbytes of extra storage each. These cartridges handle data at a much faster rate than disk drives.

In-Flight Computing
Any businessman planning to use his portable computer on an aeroplane journey may well have to choose his airline carefully. Officially, the Civil Aviation Authority (CAA) says electronic and battery operated equipment can cause interference with the aeroplanes' flight deck controls. But the airlines interpret these recommendations very differently. The German Lufthansa

airline and Australia's Qantas categorically disallow any electrical equipment on board. Pan Am, one of America's largest airlines, don't allow radios and tape recorders but do allow electronic games and computers. Whereas Japan Airlines don't mind electrical equipment of any kind, Britain's own airline, BA, prohibits computer equipment.

The less expensive portables use dedicated applications programs that are usually loaded into the computer's RAM from cassette. This is a much slower process than loading from bubble memory cartridge or disk drive. The NEC PC-8201A comes with a cassette containing several applications programs. These include a memory calculator, text formatter, investment portfolio manager, and loan evaluator. The memory calculator turns the machine into a calculator that can remember up to 99 entries. The text formatter prepares for printing files that have been entered into the word processor program by specifying margin widths, dividing the text into pages, assigning page numbers and so on. The investment portfolio manager is for use by people who want to evaluate how their stocks and shares are performing. This program analyses a portfolio of up to 50 investments, calculating losses and gains.

As with any other computer, the portable micro can be connected to peripherals such as printers, tape cassette decks, and modems. Apart from the obvious factor of size and weight, the test of a true portable is that it should be battery-powered, have its own display, and carry its word processing and communications programs in ROM.

Machines such as the Apple IIc and Apricot are advertised as portables. But they cannot be used in transit as they have to be plugged into the power supply, connected to a monitor, and have their programs loaded into RAM from disk. Apart from their smaller size and lighter weight, these have more in common with the desk-top micro than the battery-operated hand-held computer.

As well as their main batteries, portable computers are equipped with small nickel cadmium batteries that can provide emergency power. This is essential, as all data would be lost if the batteries failed without this back-up.

Most portables also have a bar code reader interface so that they can be used for stock control. The bar code reader is passed over the code on product packs. This decodes pricing and dating information that can be processed by the computer to give shopkeepers an accurate readout of their stock inventory. Of the machines that we illustrate on these pages, the Tandy Model 100, NEC PC-8201A and the Olivetti M10 are all equipped with bar code readers, but the Casio FP-200 is not. The first three machines are in fact very similar in a number of ways, as they are all manufactured at the same Japanese factory. The significant differences between them are that the Olivetti has a tilting screen, the NEC has less built-in software and the memories of the Tandy and the Olivetti cannot be expanded beyond 32 Kbytes, whereas the NEC can be expanded to 64 Kbytes. The NEC can also use interchangeable 32 Kbyte memory cartridges that retain their data even

Epson HX-20
Although it has a small screen, the HX-20 has the advantage of a built-in cassette recorder and tiny printer. A modest word processor is also included

Casio FP-200
The Casio is the cheapest of the hand-held computers but lacks the built-in word processor. Instead it offers a kind of spreadsheet

when removed from the computer. All memory upgrades for hand-helds are expensive because of the type of memory chip used.

Just as portable typewriters are used in addition to office typewriters and are not a substitute for them, so portable micros are not intended to replace the desk-top personal computer. For one thing, their small LCD displays limit their suitability for long sessions at the keyboard. The LCD display is harder to read and slower to respond to data than the cathode ray terminal.

Unlike the larger micros with their banked keyboards, the cheaper portable computers have flat keyboards that are more tiring to use. Portables at the lower end of the market cannot run popular disk-based business software.

But that said, the hand-held portable computer is here to stay. The widespread use of micros is showing more people how computer power can help them run their lives more efficiently. The portable allows them access to this power wherever they are. It cannot be long before portable computers become as common as pocket calculators.

MARCUS WILSON-SMITH

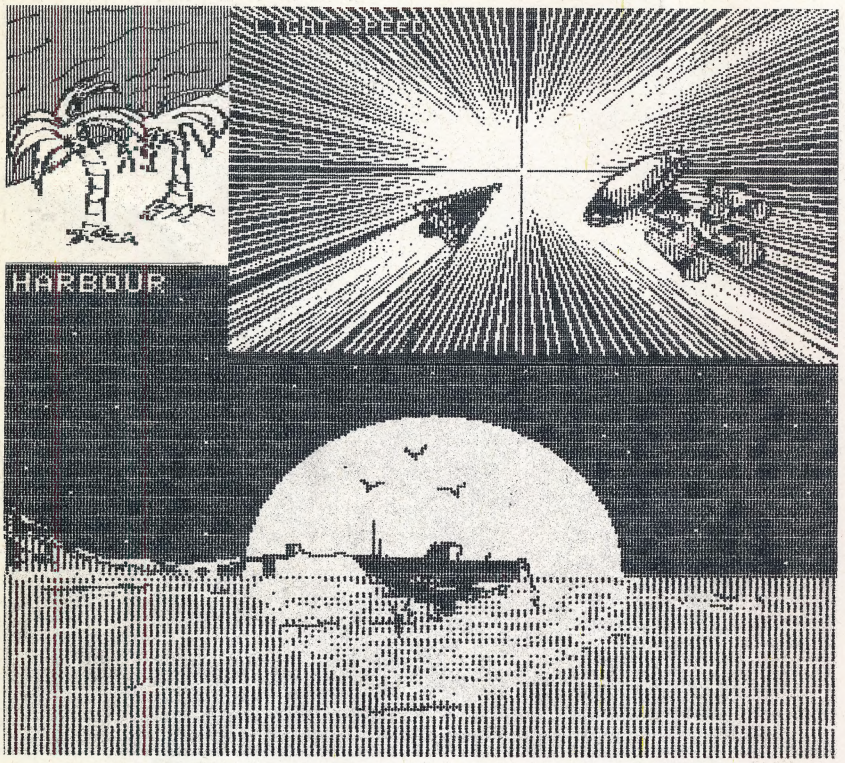
Epson PX-8
This can use CP/M-style business software, including the unbeatable Wordstar word processor that comes with the machine

Tandy TRS-80 Model 100/NEC PC-8201A/Olivetti M10
These are repackaged versions of the same micro. They share an excellent built-in word processor and a good range of interfaces. Also shown is a battery-powered modem from Olivetti

Model	Price	Standard Memory	Maximum Memory	Screen Size	Weight
Casio FP- 200	£344	8K	32K	8 × 20	1.4Kg
Epson HX-20	£599	16K	32K	4 × 20	1.8Kg
Epson PX-8	£918	64K	64K+120K*	8 ×80	2.3Kg
NEC PC-8201A	£546	16K	64K+32K*	8 × 40	1.8Kg
Olivetti M10	£495	8K	32K	8 × 40	1.8Kg
Tandy TRS-80 Model 100	£499	8K	32K	8 × 40	1.8Kg

* The NEC can take a 32K RAM cartridge and the Epson PX- 8 a 120K RAM disk.

FIRST IMPRESSIONS



SIMON DAYTON

Screen Print
These designs were drawn on-screen using a graphics tablet. The screen contents were then dumped to an Epson FX-80 printer, showing the graphic possibilities of the dot matrix printer

The graphics capabilities of dot matrix printers often tend to be overlooked, simply because users don't know they exist. In this article we show how to set up a printer to produce attractive graphics, and how to construct a screen dump program that will do the work for you.

Most home computers have a low resolution graphics mode in which pictures are built up from graphics characters, each the same size as a conventional text character. These 'block' characters have character codes greater than 127, as the numbers 0 to 127 are reserved for the ASCII character set. So PRINT CHR$(90) would print an ASCII character on the screen — 'Z' in this case — whereas PRINT CHR$(128) displays a graphic character — a black rectangle if you are using a Dragon micro.

To print the letter 'Z' on a printer, we would type LPRINT CHR$(90), so you might think that LPRINT CHR$(128) would similarly print a black rectangle on paper. Unfortunately this is not the case. This is because the characters above code 127 vary enormously between different makes of micro, and obviously printer manufacturers cannot produce a special printer for each computer on the market. What they tend to do is either copy the standard ASCII set into the codes 128 to 255, or alternatively program in their own graphics characters.

The Epson range of printers does not come with any graphics characters. Instead you can change any of the standard ASCII characters to produce your own graphics characters. This is achieved by sending suitable 'escape codes' to the printer (see pages 324 and 325).

High resolution computer graphics are constructed from small dots, or pixels, rather than from whole characters. In a similar way, high resolution printing uses small dots of ink. The print head in a dot matrix printer has a number of pins arranged in a vertical line that moves across the paper as it prints. Usually, characters are made up from a grid of dots (perhaps eight by eight dots). It is possible, however, to produce graphics by controlling the pins individually.

The first step is to switch your printer into its graphics mode. As with any other printing exercise, this is done by sending an escape code that is specific to the type of printer being used. On the Epson FX-80 for example, the necessary instructions are:

```
LPRINT CHR$ (27); "K";CHR$ (N1);(N2);
```

The letter "K" indicates graphics mode and the numbers (N1) and (N2) set the width of each line of graphics — in other words the number of dots that will fit across the page.

When in standard graphics mode, the FX-80 can print a maximum of 480 dots in one line. Other modes allow resolutions in the range of 576 to 1920 dots per line. If we wish to use the full width, therefore, 480 will be the required line length. Two numbers are required in our code to set the width, because the maximum size of each number is 255. The second number (N2) is therefore multiplied by 256 and added to the first, (N1). So for 480, the numbers are 1 and 224 (480=256×1+224). Therefore, on the Epson FX-80 printer we need the following instruction:

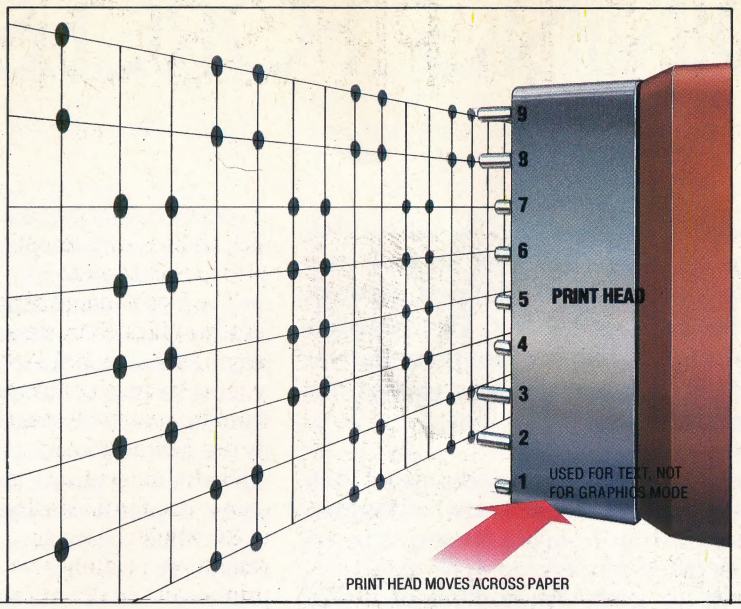
```
LPRINT CHR$ (27);"K";CHR$ (224);CHR$ (1);
```

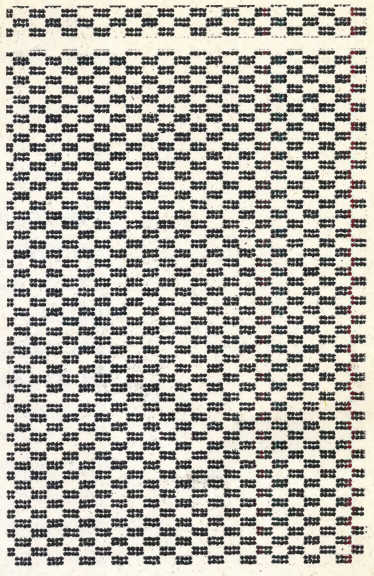
Having programmed the printer with the graphics line length we need to send the graphics data. Even though there are nine pins in the print head of an Epson FX-80, only the top eight can be used in most graphics modes. Starting from the bottom pin we number them 1,2,4,8,16,32,64 and 128. The data for all eight pins can then be represented by a single number, between 0 and 255, and this is sent to the printer using LPRINT CHR$(X), where X is the number. So if we wanted only the bottom pin to 'fire' we would send CHR$(1) to the printer; to trigger the top pin alone we would send CHR$(128). For a combination of pins we simply add up the

Pin Point
The pattern was produced on a dot matrix printer by sending alternate pairs of the decimal numbers 195 and 60 to the print head. The chart (below) shows how these numbers in binary are interpreted by the print head pins, (illustrated on the right). Controlled paper feed causes the next line to overprint the gap left by Pin 1

PIN NUMBER	PIN VALUE		
9	128	•	○
8	64	•	○
7	32	○	•
6	16	○	•
5	8	○	•
4	4	○	•
3	2	•	○
2	1	•	○
		195	60

• PIN FIRES
○ PIN DOES NOT FIRE





STEVE CROSS

numbers of each pin. This process is then repeated for each of the 480 s across the page.

In the illustration there are two pin patterns: CHR$(195) and CHR$(60). So to print the first four columns of the line pattern we type:

```
LPRINT CHR$(195);CHR$(195);CHR$(60);
    CHR$(60);
```

After four columns the pattern repeats, so a FOR...NEXT loop takes care of the rest of the line.

It is important to realise that CHR$(60) in the example does not instruct the printer to print the ASCII character with code 60 — it is a way of representing the data for the pins in the print head. The printer recognises it as such because we have previously transmitted the CHR$(27);"K" sequence to turn on the graphics mode.

This method of printing, known as *bit image printing*, is described for an Epson FX-80 printer; other printers use a similar method, but the exact details will vary. Producing graphics in this way is quite laborious, and only really suitable for patterns. A much better way of printing graphics is by means of a *screen dump*. This is a program that copies what is displayed on the monitor screen onto the paper.

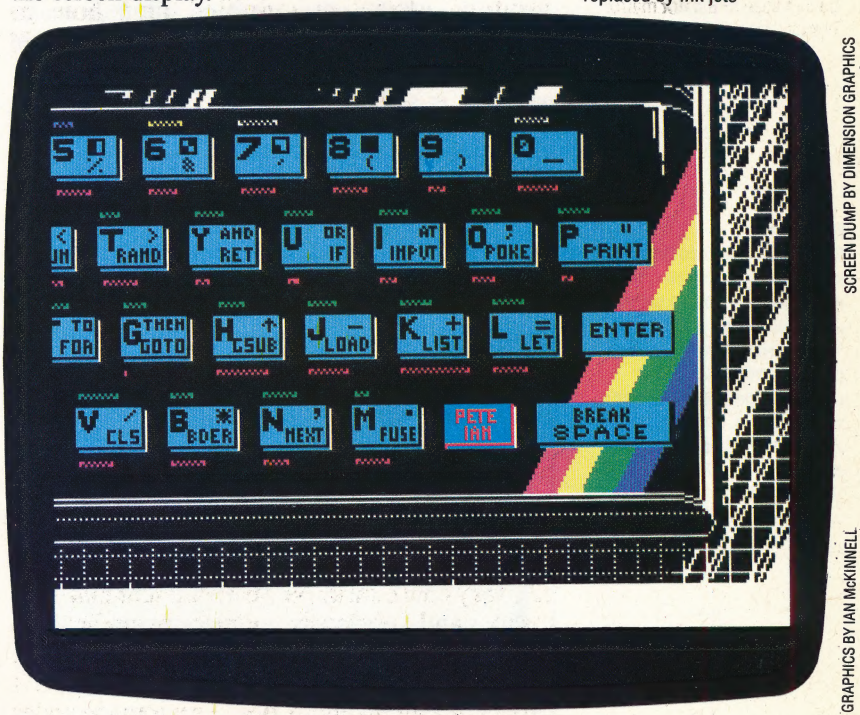
By scanning across and down the screen display, the program tests to see if the pixel is on at each position. If it is, then we want a pin in the print head to fire at the corresponding position on the paper. The scanning is done by using the POINT(x,y) function, or similar commands that are available on most micros; if a pixel is lit then the function POINT(x,y) will be 1; if it is unlit, the function is 0. The different screen resolutions of different micros mean that some adjustment might be necessary.

One problem that might have occurred to you is: how does a screen dump program handle colour displays? The usual solution is to use different dot patterns for each colour. A screen pixel that is black might be printed using four dots in the form of a square; one that is red might be represented by a two dots; and one that is white would not use any dots. The POINT(x,y) function produces a different number dependng on the colour of the pixel, and so can still be used.

Screen dump programs are usually added to the end of the program producing the picture, in the form of subroutines. To 'dump' the picture to the printer you might press the key 'P', and the program would then jump to the subroutine. A screen dump program written in BASIC tends to be quite slow, taking perhaps five minutes to print out a small picture. Machine code versions are slightly quicker.

As you can see, the graphics capabilities of dot matrix printers are reasonably advanced — if a little cumbersome to use. Once mastered however, the printed page can be as attractive as the screen display.

Splash Of Colour
This picture of the Spectrum keyboard was produced on a colour ink-jet printer — in effect this is a dot matrix printer in which the pins are replaced by ink jets



SCREEN DUMP BY DIMENSION GRAPHICS
GRAPHICS BY IAN McKINNELL

JOINING FORCES

We introduced the principles of networking on micros on page 321. Now we look at a game application for the cheapest network of all — the Sinclair ZX Net. This system is the simplest possible but its potential for both serious and amusing use should not be overlooked.

Battleships is a classic pen and paper game. Each player has two grids, which he keeps hidden from his opponent. On one he marks his own ships, on the other he marks his progress as he 'fires at' — that is, tries to guess the position of — his opponent's ships.

The program we have developed works on two Sinclair Spectrums linked together with a network. Both Spectrums must be equipped with Interface 1. Each player sits at his own screen and the two computers send each other messages that report where the players are shooting and what the results are.

The first hazard you meet is that of identifying the players. In order to communicate, each Spectrum has a different network station number. The two Spectrums start off with identical programs but somehow must end up with different station numbers. This is handled automatically by a routine at line 2000. When the program is RUN, both machines will claim to be station 1 on the public 'broadcast' channel.

Whichever machine is RUN first will become station 1 and the other machine will then make itself station 2. This system works well for Battleships. The program then assumes that whoever is station 1 is player 1 and therefore allows him to shoot first. However, if the two programs are started within a fraction of a second of each other, the two messages "I'm station number 1" simply collide and the ZX Net system will stop functioning until the players press the BREAK key.

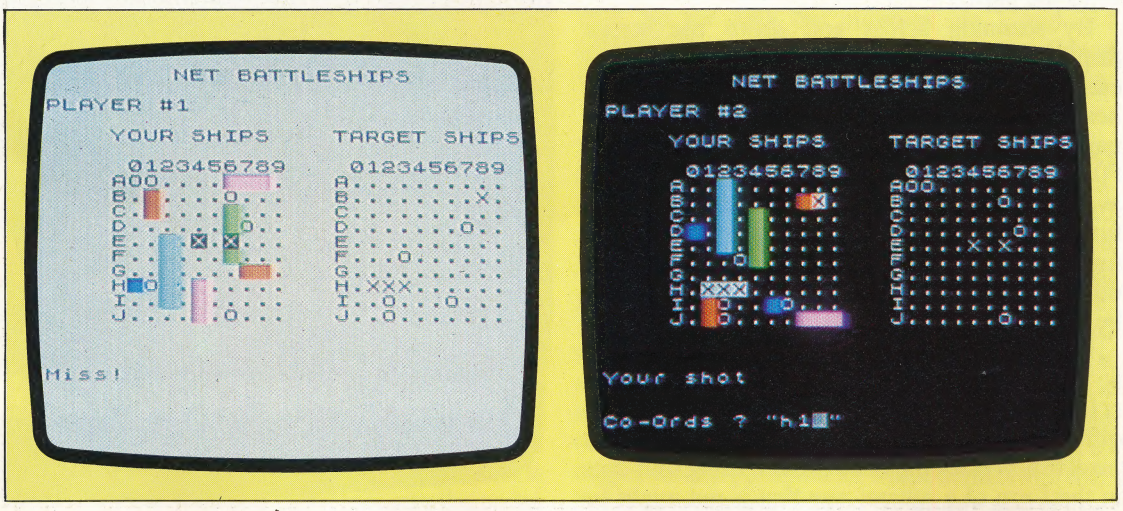
Once you know who's who, it is easy for the program to communicate with its opposite number across the network. While one player is picking a target square on his machine, the other is waiting to receive his choice. The machines will then swap over. One machine calculates the results of the shot and sends back a message while the other waits to receive the results and update its screen display accordingly.

Provided you make sure that the two programs always 'fit' together — one sending, the other receiving — this is very easy to program. You don't have to worry about timing, sending messages too late, or missing them after they've been sent because ZX Net stops until both stations are ready and then transmits the data. So it doesn't matter if one player takes a long time to select a target or if the program takes a long time to update its screen.

Another point worth noting is that the amount of data being transmitted should be kept to a minimum. There's no need to send long chunks of data. Provided both programs know what the information means, you can communicate using short codes. In Battleships, the program returns the result of a shot as a two character string. The first character is a code:

1 Miss
2 Hit a ship
3 Hit and sunk a ship
4 Hit and sunk a ship and won the game

The second character is the class of ship that was hit (or a 0 for a miss). The program at the other end can decode this information and display appropriate messages. This method makes the time taken to execute each turn so fast you wouldn't think that another computer was involved at all.



LIZ HEANEY

Fleet Action
Each player's fleet comprises ships of different sizes (from motor torpedo boat to carrier) placed anywhere in the grid. The screen displays the position and status of one player's ships and the enemy's shots at them, and the position and effect of his shots at the enemy's ships: "X" shows a hit, "O" for a miss

Playing The Game

This program requires a slightly complex starting procedure because it runs separately on two Spectrums. A copy of the program must be loaded into each computer.

Both may be loaded from cassettes but it is much quicker to load one Spectrum from cassette (or Microdrive) and then transmit the program across the network to the other machine.

To do this, type LOAD *"n";0 at the receiving end and SAVE *"n";0 on the machine that has the program in memory. Next, the players should decide who is to shoot first. This player should RUN the program slightly before the second player. The program then assigns network numbers to both machines and works out which copy of the program is playing first and which second.

When the program begins, both players must set up the positions of their ships. This is done by specifying the location of one end of each craft on the 10 × 10 playing grid and saying whether the rest of the ship is up, down, left or right of that position. This sounds complicated but is convenient in practice. Each player has two MTBs (length 1 square), two Cruisers, (length 2 squares), two Battleships (length 3), a Destroyer (4) and an Aircraft Carrier (5).

The players then take turns to shoot at a square on each other's grid, and the program evaluates the result of each shot. A win is achieved by destroying all of one player's ships. Both players should type RUN to play again, remembering that whoever wants to start should RUN first

```
  10  REM Networked Battleships Game
  11  REM
  12  REM 2 Spectrums,Interface 1s
  13  REM June 84/Version 1.6
  15  REM *** init everything
  30  GO SUB 2000: REM fight for net
  40  LET s$="                                ":
 REM 34 spaces
  50  DIM s(8): REM ship types
  60  DIM n$(8,12): REM ship names
  70  DIM a(10,10): REM squared paper!
  80  FOR i=1 TO 8: READ s(i),n$(i): NEXT i
  90  DATA 1,"MTB",1,"MTB",2,"Cruiser",2,"Cruiser"
 100  DATA 3,"Battleship",3,"Battleship",4,"Destro
yer",5,"Carrier"
 110  LET sc=8
 120  GO SUB 3000: REM init screen
 130  GO SUB 4000: REM set up ships
 140  REM now jump depending which
 150  REM player we are.  #1 shoots first
 160  IF sta=2 THEN  GO TO 400
 200  REM *** Take a shot!
 210  LET m$="Your shot": GO SUB 6000
 220  GO SUB 7000: IF e=1 THEN  GO TO 210
 230  OPEN #4;"n";him
 240  PRINT #4;a$
 250  CLOSE #4
 260  REM wait & get results back
 270  OPEN #4;"n";him
 280  INPUT #4;a$
 290  CLOSE #4
 300  LET r=VAL (a$(1 TO 1)): LET x=VAL (a$(2 TO ))
 310  IF r=1 THEN  LET m$="Miss!": PRINT AT 6+q,18
+p;"O";: GO SUB 6000
 320  IF r>1 THEN  LET m$="Hit!": PRINT AT 6+q,18+
p;"X";: GO SUB 6000
 330  IF r>2 THEN  LET m$="You've sunk an enemy "+
n$(x): GO SUB 6000
 340  IF r=4 THEN  LET m$="Congratulations .... Yo
u win!!!": GO SUB 6000: STOP
 400  REM *** Enemy fire
 410  LET m$="Enemy firing": GO SUB 6000
 420  OPEN #4;"n";him
 430  INPUT #4;a$
 440  CLOSE #4
 450  LET p=VAL (a$(2 TO ))+1: LET q=CODE (a$)-64
 460  LET m$="Enemy firing at "+a$: GO SUB 6000
 470  LET x=a(p,q): LET a(p,q)=0
 480  IF x=0 THEN  LET r=1: GO TO 530
 490  LET r=2
 500  LET s(x)=s(x)-1
 510  IF s(x)=0 THEN  LET r=3: LET sc=sc-1
 520  IF sc=0 THEN  LET r=4
 530  LET a$=STR$ (r)+STR$ (x)
 540  OPEN #4;"n";him
 550  PRINT #4;a$
 560  CLOSE #4
 570  IF r=1 THEN  LET m$="It's a miss": PRINT AT
6+q,4+p;"O";
 580  IF r>1 THEN  LET m$=n$(x)+" damaged": PRINT
AT 6+q,4+p;"X";
 585  IF r>2 THEN  LET m$=m$+" and sunk"
 587  GO SUB 6000
 590  IF r=4 THEN  LET m$="Sorry .... you lose": G
O SUB 6000: STOP
 600  GO TO 210
2000  REM *** decide who's who
2005  CLOSE #4
2010  OPEN #4;"n";0
2020  PRINT #4;"1"
2030  CLOSE #4
2040  OPEN #4;"n";0
2045  INPUT #4;a$
2050  CLOSE #4
2060  IF a$="1" THEN  OPEN #4;"n";0: PAUSE 5:
PRINT #4;"2": LET sta=1
2070  IF a$="2" THEN  LET sta=2
2080  CLOSE #4
2090  FORMAT "n";sta: LET him=3-sta: RETURN
3000  REM *** set up screen
3010  LET col=0: IF sta=2 THEN  LET col=7
3020  PRINT ;:  BORDER 7-col: PAPER 7-col: INK col
: CLS
3030  PRINT TAB 8;"NET BATTLESHIPS"
3040  PRINT : PRINT "PLAYER #";sta
3050  PRINT : PRINT "     YOUR SHIPS     TARGET SHIP
S"
3060  PRINT : PRINT "      0123456789     0123456789
"
3070  FOR i=1 TO 10
3080  PRINT "     ";CHR$ (i+64);".........    ";CHR
$ (i+64);"........."
3090  NEXT i
3100  RETURN
4000  REM *** Set up ships
4010  LET m$="Please position your ships": GO SUB
6000
4020  FOR s=1 TO sc
4030  LET m$=STR$ s+". "+n$(s)+"length "+STR$ (s(s
)): GO SUB 6000
4050  GO SUB 7000: IF e=1 THEN  GO TO 4030
4055  IF s(s)=1 THEN  LET xd=0: LET yd=0: GO TO 41
30
4070  INPUT "U, D, L or R ? ";a$: LET xd=3: LET yd
=3
4080  IF a$="U"  OR a$="u" THEN  LET xd=0: LET yd=
-1
4090  IF a$="D"  OR a$="d" THEN  LET xd=0: LET yd=
1
4100  IF a$="L"  OR a$="l" THEN  LET xd=-1: LET yd
=0
4110  IF a$="R"  OR a$="r" THEN  LET xd=1: LET yd=
0
4120  IF xd=3 AND yd=3 THEN  GO TO 4070
4130  LET l=s(s): LET x=p: LET y=q
4140  IF x<1 OR x>10 OR y<1 OR y>10 THEN  LET m$="
Move the ship away from the edge": GO SUB 6000: GO
 TO 4030
4150 IF a(x,y)<>0 THEN  LET m$="Please reposition
the ship": GO SUB 6000: GO TO 4030
4160  LET x=x+xd: LET y=y+yd
4170  LET l=l-1: IF l>0 THEN  GO TO 4140
4180  LET l=s(s): LET x=p: LET y=q
4190  LET a(x,y)=s: INK s(s): PRINT AT 6+y,4+x;" "
;: INK col
4200  LET x=x+xd: LET y=y+yd
4210  LET l=l-1: IF l>0 THEN  GO TO 4190
4220  NEXT s
4230  LET m$="Prepare for action!!!": GO SUB 6000
4240  RETURN
6000  REM *** Print m$
6010  PRINT AT 20,0;s$;AT 20,0;m$: PAUSE 100: RETU
RN
7000  REM *** Validate co-ords
7010  LET e=0
7015  INPUT "Co-Ords ? ";a$
7020  IF LEN a$<>2 THEN  LET e=1: GO TO 7100
7030  FOR i=1 TO 2
7040  LET c=CODE (a$(i TO i)): IF c>=97 AND c<=122
 THEN  LET a$(i TO i)=CHR$ (c-32)
7050  NEXT i
7060  LET q=CODE (a$(1 TO 1)): LET p=CODE (a$(2 TO
 2))
7070  IF q<65 OR q>74 THEN  LET x=q: LET q=p: LET
p=x
7080  IF q<65 OR q>74 THEN  LET e=1
7090  IF p<48 OR p>57 THEN  LET e=1
7100  IF e=1 THEN  LET m$="Please re-enter co-ordi
nates": GO SUB 6000: RETURN
7110  LET q=q-64: LET p=p-47
7120  RETURN
```

CONTROL CHARACTER

Control characters are ASCII codes that are used to indicate to the computer that a specific operation must be carried out. These codes fall outside the range used for normal text, numbers and punctuation, and are often not printable. Many control characters are universally recognised — ASCII 13 is always used to indicate a Carriage Return — while others are used for different purposes on different machines. The Oric and Atmos computers, for example, use control characters for double-height printing, for disabling the screen and for changing the displayed colours. Word processing packages use control characters for formatting text output by sending instructions to the printer.

COURSEWARE

You may be getting a little wary of the seemingly endless series of 'ware' words. The terms *hardware* and *software* are by now universally recognised; *firmware* denotes system software that is held in ROM; *courseware* is used in educational circles in connection with computer aided instruction (see page 235). It refers to the range of educational software that is available for a particular machine. A microcomputer's courseware may consist of a selection of independent programs from different sources that cover separate topics and that have nothing in common. The term is also applied to programs from the same source that use the same command structure but cover different subjects in the curriculum. Sometimes the word is used to refer to a single program, with demonstration and questions held in separate data files.

CP/M

Like Hoover, Biro or Thermos, CP/M is a proprietary product name that has found its way into common usage as a generic term and is often used as a description of a business computer. CP/M is, in fact, an operating system that was written by Gary Kildall of Digital Research for 8080- and Z80-based machines. CP/M stands for Control Program/Monitor (or Control Program for Microprocessors), and was the first operating system to be adopted by a large number of computer manufacturers, enabling software writers to develop programs that would run on many different machines.

Its wide acceptance was due to the fact that it was the first in the field, and in many respects CP/M is a far from ideal system, containing several hang-overs from the crude microcomputer system for which it was written. CP/M file-editing, for example, is based on a facility for editing reels of punched paper tape. Numerous versions of CP/M have been released. These include Concurrent CP/M, which allows several different programs to be run simultaneously, and Personal CP/M, which is stored in ROM instead of on disk.

CPU

The CPU, or *central processing unit*, is the part of the computer that does all the important work. All computers, irrespective of size, have a CPU. The CPU on a microcomputer, however, is contained in a single silicon chip, otherwise known as a microprocessor.

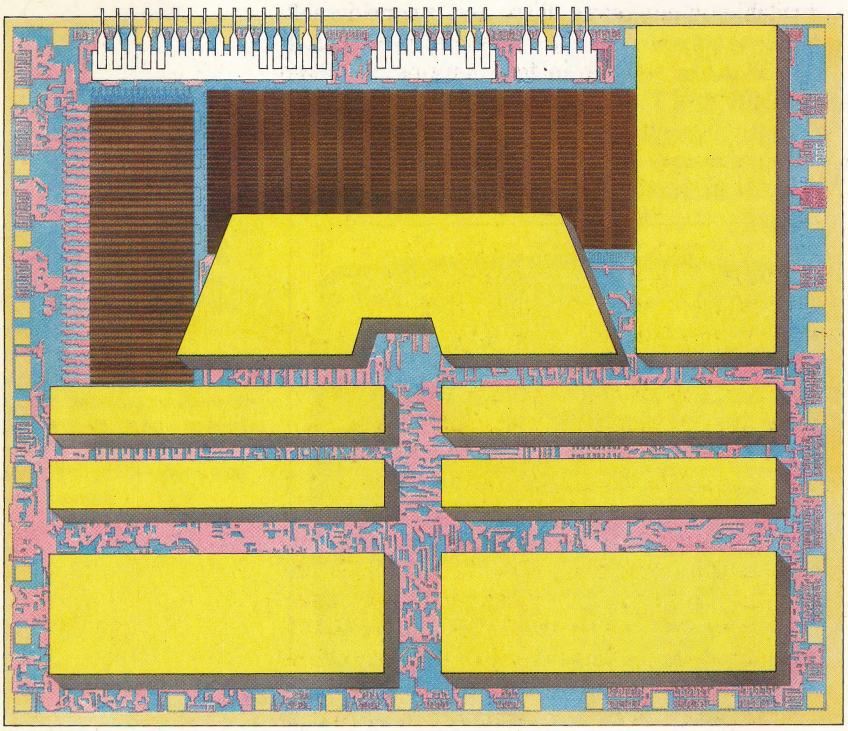
The task of the CPU is to receive program instructions in machine code and carry them out. These instructions may be used to move numbers around in memory or to perform simple arithmetic. All instructions and numbers are in the binary number system.

Each CPU is made up of many thousands of components, although some CPUs are more complex than others. One measure of the complexity of a CPU is the size of numbers that it can handle, measured in binary digits (bits). Many calculators use four-bit CPUs, home micros generally use eight-bit CPUs, some business micros use 16-bit CPUs, and mini and mainframe computers often use 32 or 64-bit CPUs.

Some microprocessor CPUs combine more functions on one chip than others. Many have built-in clocks to produce the vital timing signals, others need a separate clock chip. A number of CPUs include small amounts of memory, while others have all their main memory outside the CPU. The CPU handles all the real computing work, but needs external components before it can be used.

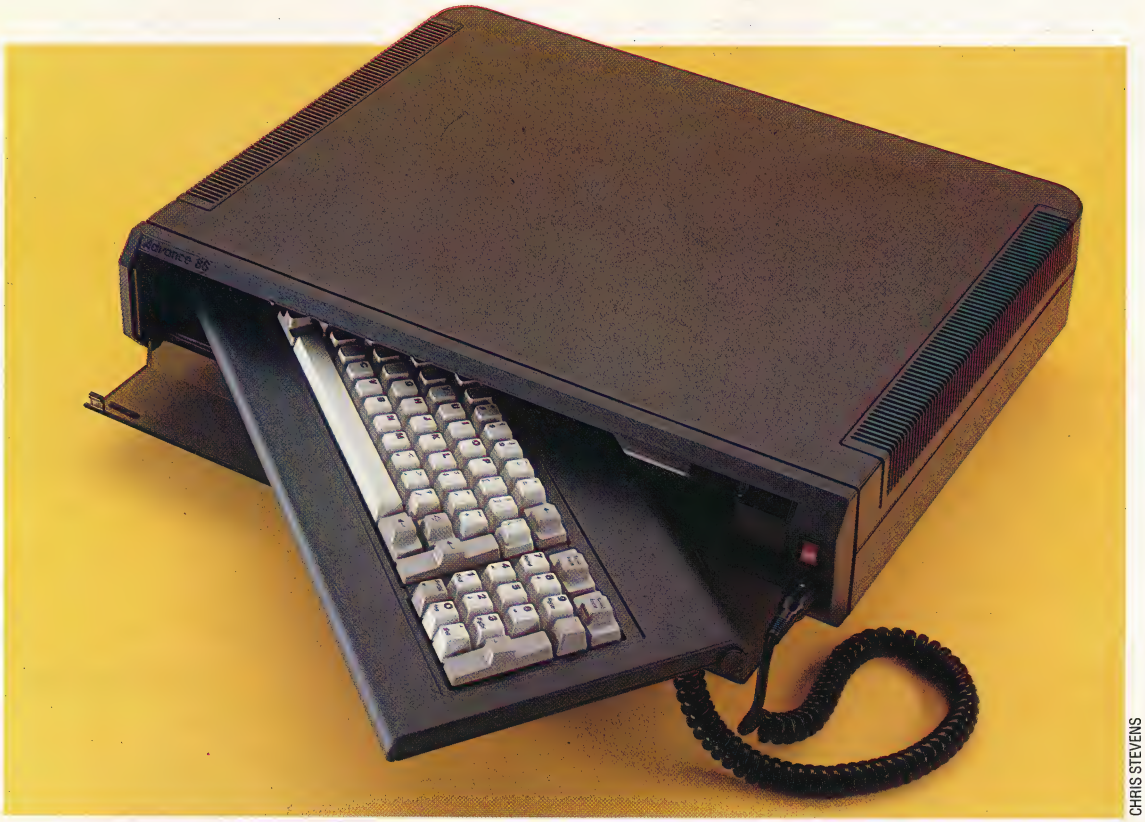
The CPU sends and receives data and control signals over three sets of signal lines, known as buses. These are the data, control and address buses. These buses, plus a few other connections (such as power), enter and leave microprocessor CPUs through the short 'legs' that emerge from the package. The CPU chip itself is encased in this plastic or ceramic package and measures only 5mm (1/5th of an inch) in diameter.

CPU
A typical micro CPU showing pin connectors for the buses. The other shapes represent the ALU, registers, program counter, stack pointer, and control block



KEVIN JONES

ADVANCED STUDIES



CHRIS STEVENS

Advance 86a
The Advance 86 is sold in two versions — the £399 Advance 86a, pictured here, which is marketed as a home computer, and the £1,499 Advance 86b, an IBM-compatible business micro

Microcomputer manufacturers have for years advertised their wares with claims that cheap home machines may — once their owners have familiarised themselves with their operation — be upgraded to full business machines. Advance Technology is one of the first companies to fulfil that promise with the Advance 86 micro.

The Advance 86 is supplied in two configurations: the 86a is a cassette-based home microcomputer that is priced at BBC Micro level but boasts a full 128 Kbytes of RAM; while the 86b is a similar machine augmented by twin $5\frac{1}{4}$ inch disk drives and a more extensive BASIC. 86a owners may upgrade to the 86b by simply clipping on an 'extension package' containing the drives. The result is a fast IBM-compatible business machine at half the price of the IBM PC.

The 86b is manufactured in two parts — the keyboard and the microprocessor box. The latter is considerably larger than it needs to be, measuring 520mm × 400mm × 95mm (21 × 16 × 4 inches), and is cased in black plastic with a hi-fi-style smoked perspex door that allows the keyboard to be stored inside. All sockets are mounted in this unit, and the power transformer is located inside it.

The keyboard is connected to the processor unit by means of a coaxial cable and five-pin DIN socket, above which is the on/off switch with an LED indicator. The keyboard is certainly the best available on any home micro and is considerably better than those on most business machines. It has 84 keys arranged in three groups: the main alphanumeric keys, a set of 10 function keys, and a numeric keypad that doubles as a very comprehensive set of cursor controls.

The function keys are set up to provide some of the more commonly used functions — RUN, LIST, SAVE, LOAD, etc. These functions may easily be altered; each key may be allocated a string of up to 15 characters to identify the command and the bottom line of the screen displays a six-letter label for each one. The numeric keypad is controlled by a toggle switch marked 'Num Lock'. In normal use the pad acts like a calculator, but pressing Num Lock gives an entirely different effect — the keys then control the cursor, allowing it to be moved around the screen with precision and speed.

Inside the microprocessor box is a relatively small circuit board containing the Intel 8086 16-bit microprocessor (compatible with, but faster than, the IBM's 8088) and 128 Kbytes of RAM. Also fitted are sockets that allow the RAM to be doubled, although memory available to BASIC is limited to 62 Kbytes. This is hardly a drawback, as

Clever Keyboard
The IBM Personal Computer keyboard is claimed as being the best available on any microcomputer. The Advance keyboard closely mimics IBM design but some keys are repositioned and the Advance, rather strangely, has two Return keys

this amount is more than enough for most applications.

The Advance is well supplied with sockets and interfaces. These include a 'mains out' socket that allows a television or monitor to be run from the computer's power supply; a socket enabling a television to be used as a display; 'comp sync' and RGB outlets for either composite video or RGB (red, green, blue) monitors; a standard Centronics interface for connection of a parallel printer; two joystick ports; and a five-pin DIN socket for tape recorder use. If the upgrade is fitted, an RS232 socket is available for connection of a serial printer or modem. All interfaces take IBM-style leads.

In text mode, the Advance displays either 25 lines of 40 characters or an IBM-like 80 × 25 screen; the latter is barely readable unless a monitor is used. The bottom line of the screen normally displays function key labels, but these may be turned off and the full screen made available. In this mode, 16 colours (either steady or flashing) can be used. The medium resolution screen supports four colours, with a graphics display of 320 × 200 pixels or text in 40 × 25 format. The high resolution mode gives a black and white display of 640 × 200 pixels or 80 × 25 text. Seven full 40 × 25 screens may be stored in RAM and recalled instantly — very handy for menus, help pages, etc. In 80-column mode, four screens may be stored and recalled. Despite the large number of available colours, there are some annoying restrictions placed on their use. In text mode the background is limited to one of eight colours, although foreground and border may use any of the full set. When using the medium resolution mode, things become a little more complicated. Although four colours may be displayed, these may not be chosen from the full range; instead one of two set groups (or 'palettes') must be selected.

Graphics commands on the 86a are limited to PSET (which sets the colour of an individual screen pixel) and LINE (a fast line-drawing command that also gives boxes). More useful commands like CIRCLE, PAINT (for filling any on-screen shape with colour), DRAW (which allows any shape to be defined and drawn) and GET and PUT (enabling areas of the graphics screen to be copied into arrays, then returned to the display in different colours or sizes) are supplied only on the 86b's Disk BASIC. This is a pity, as the one thing a home computer owner is likely to require is a full set of graphics commands. The Advance character set includes the normal ASCII range, together with mathematical symbols and block graphics featuring playing card suits, music notes, Greek letters and even one of those hideous smiling faces that you often see on badges and stickers. User-defined characters may also be created.

The rest of the Advance BASIC is almost beyond reproach. Although lacking some of the 'structured' facilities of BBC BASIC, it is fast and very easy to use. Useful features include automatic line numbering and renumbering, PRINT USING, which allows easy formatting of screen displays, and the SWAP command that allows the values of two variables to be exchanged. Sound facilities are good, if not startling, but again the Disk BASIC is required for the full set of commands. Cassette operation is straightforward — BASIC and machine code programs are loaded with a simple LOAD command, and programs will run automatically after loading if the letter 'R' is appended to the instruction.

One of the most impressive features of the Advance is the excellent screen editor. By using the Num Lock key to set up the numeric keypad as cursor controls, the user may move the cursor freely about the screen, making corrections and insertions at any point.

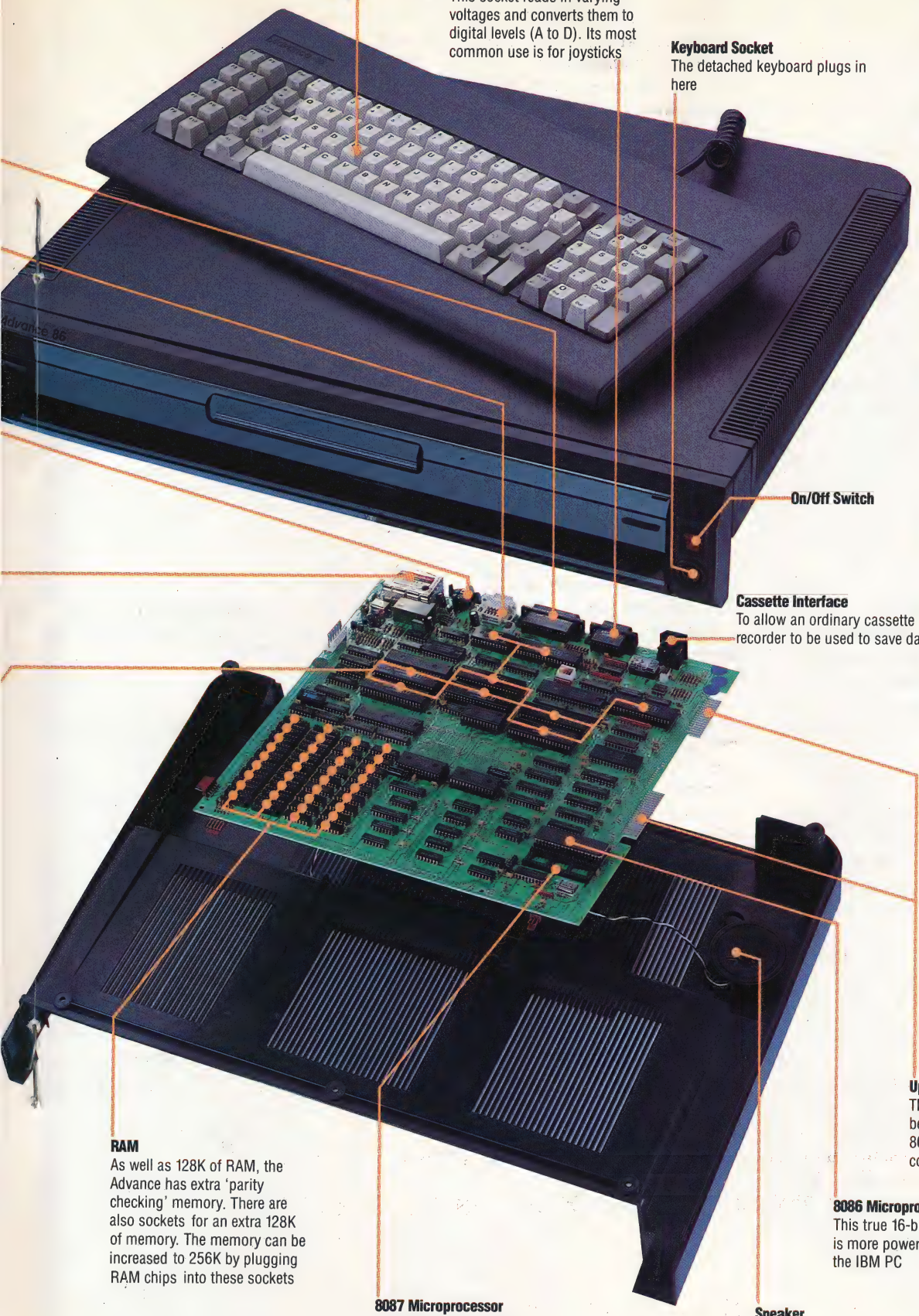
All in all, the Advance does seem to live up to its promise of providing a home computer that may be upgraded to full business status. Undoubtedly the facilities offered by the 86b's more comprehensive Disk BASIC are considerably superior to those on the cheaper machine, but the 86a can certainly bear comparison with any home micro currently available. The BASIC may not be up to BBC standard, but the Advance's superb keyboard and huge memory make it a more attractive proposition at the same price.

Centronics Printer Interface
RGB Output
A high quality colour monitor can also be used
Composite Video Output
This enables a monochrome or colour monitor to be used
TV Modulator And Output
To allow an ordinary TV set to be used
Ferranti ULA Chips
The cost of the computer is kept down by combining many circuits into nine specially designed ULA (uncommitted logic array) chips

Advance To IBM

The Advance's main attraction is that it may be bought as a home computer and then upgraded to full business specification.

The expanded system is almost totally IBM-compatible; our picture shows the Advance running Microsoft's Flight Simulator, which is regarded as the ultimate test of compatibility. The expansion box costs £1,100 and fits on top of the 86a's main unit. It should be fitted by a dealer — W. H. Smith markets the computer in the U.K.



IBM-Style Keyboard

Joystick And A To D Convertor
This socket reads in varying voltages and converts them to digital levels (A to D). Its most common use is for joysticks

Keyboard Socket
The detached keyboard plugs in here

On/Off Switch

Cassette Interface
To allow an ordinary cassette recorder to be used to save data

RAM
As well as 128K of RAM, the Advance has extra 'parity checking' memory. There are also sockets for an extra 128K of memory. The memory can be increased to 256K by plugging RAM chips into these sockets

8087 Microprocessor
The high speed maths processor can be added here

Speaker

Upgrade Connectors
The Advance 86a upgrades to become the IBM-compatible 86b by attaching a unit to these connectors

8086 Microprocessor
This true 16-bit microprocessor is more powerful than that of the IBM PC

ADVANCE 86A

PRICE
Advance 86a £399
Advance 86b £1,499

DIMENSIONS
95 × 400 × 520mm

CPU
Intel 8086, 4.77 MHz

MEMORY
128K RAM (expandable to 256K), 64K ROM

SCREEN
25 rows of 40 columns or 25 rows of 80 columns of text, graphics 320 × 200 (4 colours) or 640 × 200 (black and white). 16 colours in text mode

INTERFACES
RGB and composite video monitors, joysticks (2), Centronics parallel printer interface, cassette port, 'mains out' socket. RS232 port (86b only)

LANGUAGES AVAILABLE
BASIC in ROM (86a), Disk BASIC (86b)

KEYBOARD
Typewriter-style, 84 keys, including 10 function keys and numeric keypad

DOCUMENTATION
User guide supplied, BASIC manual also available. Adequate but lacking in detail

STRENGTHS
Excellent keyboard and screen editor. Disk BASIC very comprehensive, and larger memory than any home machine. IBM compatibility means considerable amount of software available

WEAKNESSES
ROM BASIC lacking in commands to make best use of sound and graphics. Limitations in colours available in graphics modes

CHRIS STEVENS

LANDING CRAFT

In this instalment of our series of short, entertaining programs we look at one of the oldest games in computing: Lunar Lander. In many ways it may seem too simple when compared with the fast, noisy excitement of today's arcade games. But, properly programmed, the game is quite subtle and difficult to master.

In Lunar Lander you have to guide a lunar landing craft onto the surface of the moon (or some other planet). The ship's on-board computer has failed, so you have to bring the craft down carefully with short bursts of its rocket engine. Obviously, you must hit the surface at a reasonable speed and the game involves a careful balance between dropping too fast and using up your limited amount of fuel high above the planet's surface.

The essential element of the game is that it is actually a simulation. If you program it so that firing the rocket for two seconds always takes 10 km/h from its downward speed, then the game is too easily mastered. The idea is for the program to reflect the true behaviour of a spacecraft in such conditions as accurately as possible. Obviously, the mathematics to do this is very complicated, so the program given here is a simplified version. However, it does have many of the features of a true simulation.

Let's take a closer look at the lander problem:

- The planet you are descending on has a certain gravity. This will cause the spacecraft to accelerate towards the planet as it lands.
- The spacecraft has a rocket engine that will counteract the effects of gravity by pushing the craft upwards.
- The spacecraft has a mass (or weight). The greater this is, the less effect the rocket motor will have on pushing the craft upwards. The mass of the spacecraft is made up of its own weight and the weight of the fuel inside it. As the fuel is used up, the spacecraft becomes lighter.

So to reflect the way the lander behaves, we need a set of equations involving acceleration, mass,

Planet Pull
The figures shown give the approximate gravitational pull of the Sun, Moon and the planets in our solar system in metres per second per second. These values for the variable g in the program should be entered in line 20. There are, of course, certain bodies that it would be foolish or deadly to land on (such as the Sun, Jupiter and Earth) but you might like to ignore such matters for the purpose of the game

ADRIAN MORGAN

velocity and so on. These can be very simple or very complicated, depending on how detailed and accurate we wish to be. For the purpose of this game, we've kept them fairly simple.

The main thing we need to know is the height of the spacecraft. Obviously, it is continually moving, either dropping due to gravity or accelerating away from the planet because of overuse of the rocket engine. To allow us to work out where it is at any one moment, we divide 'time' up into a series of steps or periods.

In each period, we can calculate how far the craft has moved, what the change is in its speed and mass and so on. These periods can be any length you like — the shorter they are, the more accurate the simulation. Once we introduce the idea of periods, writing the equations is easy.

Speed is measured as so many units per hour. In two hours, a car travelling at 10 km/h (10 kilometres per hour) will move 20 kilometres. In three hours, it will move 30 kilometres, and so on. This gives the formula:

Distance = Time × Speed

So in each period we can calculate how far up or down the lander moves by multiplying its velocity by the length of the period (which we define as unity). We can then adjust the velocity by accelerating the craft by the planet's gravitational pull and decelerating it by any push from the rocket motors.

Acceleration due to gravity is always constant (the variable g in the program) and will depend on the planet that is being approached. The illustration shows the values for the planets in our solar system but you could experiment with other values or have the program generate g randomly to provide a more difficult game.

Simulating the rocket motor is slightly more complicated. In this version, the player can burn from one to nine units of fuel in a given period, and the program calculates the resulting acceleration, taking into account the mass of the craft. The exact formula depends on the power of the rocket engines and the type of fuel used. In this program, the figures are chosen to make the game hard to beat; you could try altering them to see how they affect play.

One twist that can be added to the simulation is that it should be played in *real time*. This is a much abused phrase and nowadays usually means that the program (game, simulation, etc.) runs continuously, and never stops for the user to enter data or commands. This is often only the difference between using an INPUT command to gather information and an INKEY$ or GET.

Obviously, Lunar Lander is a better game if the program doesn't come to a halt to calculate how much rocket fuel to burn. If it did this, the player would have time to consider the situation, make a few calculations, and so on. In real life, it would be more a question of quick thinking.

In our lander program, we have a loop that operates once for each time interval in the program. By adjusting the period so that it is actually the time taken to execute the loop, the simulation operates in real time. It takes as long to land the spacecraft in the simulation as it would to land it in real life. Although this is desirable in a simulation, it can be quite hard to get right in a game like this. Usually, the simulation takes too long to be interesting as a game.

There are many things you can do to the basic lander program. The most obvious is to add a graphic display of the descent. Ideas for this vary from a simple round altimeter dial to a side view of a little ship, or even a scaled view downwards of the landing pad. You could also try adding sideways motion so that the craft has to be positioned over the pad as well as lowered onto it.

The craft won't normally drift sideways in space because there is nothing to push or pull it in any other direction than down. But if you're landing on a planet with an atmosphere, you could add in the problem of a surface wind and so on. Some sophisticated versions of the program have several landing areas, situated at the bottom of tunnels and craters so that landing requires plenty of careful steering.

Lunar Lander may seem old hat compared to the fast and furious arcade games of today. But programming and playing it is many people's first encounter with a computer simulation and the whole complex field of making computer programs reflect objects and events in the real world. Programming a moon landing can be the first step to a whole new range of programming projects.

Touchdown

```
  10  REM Lunar Lander Game
  20  LET g=-1.6
  30  LET t=1
  40  LET f=1000
  50  LET v=0
  60  LET h=2000
  70  LET m=2000+f
  80  LET g=g*t
 100  REM Update screen
 110  PRINT AT 0,0
 120  PRINT "          Lunar Lander"
 130  PRINT : PRINT "Height....";INT h;"   "
 140  PRINT : PRINT "Speed.....";INT v;"   "
 150  PRINT : PRINT "Fuel......";f;"   "
 160  PRINT
 165  IF h<0 THEN  GO TO 400
 170  IF f<=0  THEN  LET f=0: PRINT "*** OUT
OF FUEL      ": GO TO 190
 180  PRINT "Key rocket burn 0-9 "
 190  LET b=0: IF f>0 THEN  LET a$=INKEY$: IF
 a$<>"" THEN  LET b=VAL a$
 200  IF b>f THEN  LET b=0
 210  LET h=h+v*t
 220  LET v=v+g
 230  LET v=v+(b*3000)/m
 240  LET f=f-b: LET m=m-b
 250  FOR i=1 TO 50: NEXT i
 300  GO TO 110
 400  REM On planet surface
 410  IF v>-10 THEN  PRINT "*** Safe Landing
... Well done": GO TO 500
 420  IF v>-20 THEN  PRINT "*** CRUNCH! ... y
ou wrecked the lander but the crew survived!"
: GO TO 500
 430  PRINT "*** SMASH!  ... Lander destroyed
... no survivors"
 440  PRINT : PRINT "You've just blasted a ne
w       crater ";INT (-v*2.1);" Km wide"
 500  PRINT : PRINT "Play again (Y/N) ? ";
 510  LET a$=INKEY$: IF a$="" THEN  GO TO 510
 520  IF a$="y"  OR a$="Y" THEN  RUN
 530  IF a$<>"n" AND a$<>"N"  THEN  GO TO 510
 540  CLS : STOP
```

Basic Flavours

This is a Spectrum listing: other machines do not require use of the word LET. On the BBC Micro, replace line 110 by:

```
10 PRINT TAB(0,0)
```

Replace INKEY$ in lines 190 and 510 by INKEY$(0). Replace VAL a$ in line 190 by VAL(a$). Replace INT h and INT v in lines 130 and 140 by INT(h) and INT(v).

On the Commodore 64 and Vic-20, replace line 110 by:

```
110 PRINT CHR$(19)
```

Replace LET a$=INKEY$ in lines 190 and 510 by GET a$. Replace VAL a$ in line 190 by VAL(a$). Replace INT h and INT v in lines 130 and 140 by INT(h) and INT(v).

On the Oric/Atmos replace line 110 by:

```
110 PRINT @ 0,0
```

Replace INKEY$ in line 190 and 510 by KEY$. Replace VAL a$ in line 190 by VAL(a$). Replace INT h and INT v in lines 130 and 140 by INT(h) and INT(v)

STYLE COUNSEL

Documenting a program involves much more than adding comments to it or writing user instructions. A well-documented program has sufficient information to indicate what it is meant to be doing, and how it is meant to be doing it. We show how a simple program, in BASIC and PASCAL, can be given suitable documentation.

Consider the first version of our program (Listing 1). It is clearly a great mystery; what it does is anyone's guess. Apart from saying that it 'inputs two numbers, multiplies them with two other numbers, adds the two results together and prints the answer', there is very little evidence of the precise task that the code performs. Now look at the second version of the program (Listing 2). All is revealed. Yet no comments have been added, no program titles or REMark lines inserted and no external documents have been produced.

It is worth taking a detailed look at the differences between these two versions. First of all, the meaningless numbers of the first listing have been replaced by names (AYEAR and AMONTH). Numbers whose values do not change while the program is running are called *constants*. Some languages, such as PASCAL, have a special notation for constants (in Listing 2, the two constants are defined separately from the variables), while other languages, like BASIC, do not. (Lines 10 and 20 of the BASIC program use variables to define the constants.) Giving names to constants is really only worthwhile if they are going to be used frequently, otherwise comments in the program would serve the purpose just as well.

The second crucial difference is that all the confusing variable names have been given longer, intelligible names. The ones that we chose here (NYEARS to replace A, ageinsecs instead of e, and so on) were picked because they are each less than 10 characters long, and the first two characters distinguish them from each other. The reason for this last requirement is explained shortly.

Generally, it is good practice to give your variables names that are related to the role they play in the program. For example, you could call a loop counter LOOP (instead of the usual J or I), and the first and last values of the counter could be put into constants or variables with appropriate names. Thus, a loop reading like this:

```
FOR J = 1 TO 10 . . . NEXT J
```

could look like this:

```
FOR LOOP = FIRST TO TENTH . . . NEXT LOOP
```

Long variable names do, of course, take longer to type in and use up more memory, but they do have the advantage of making programs easier to understand and speed up the debugging process. If your language uses only the first two characters of a name to distinguish between them, make sure that the names you choose differ in the first two characters. Thus, two long variable names (say CODENO and COMP) may look different to the programmer, but be indistinguishable to the computer.

Another major difference between the listings is that the second uses long and meaningful prompts for its input and adds a sensible explanation to its output (the PRINT lines in BASIC, the write lines in PASCAL). This achieves two very important things. The first is that it makes the program more readable. Even if the variables were single letters, the program would still make a lot more sense than previously. The second, and more important, benefit is that it makes the program accessible, even to someone who has never seen it before.

PROGRAM LAYOUT

PASCAL users will already be aware of the advantages of laying out a program neatly on the screen. Very simple features — like indenting lines, leaving blank lines and using a mixture of upper and lower case — can turn an impenetrable mass of symbols into a tidy and legible piece of logic. Formatting a program for the screen or the printer really comes into its own when your programs use loop constructs (FOR . . . NEXT, WHILE . . . WEND, REPEAT . . . UNTIL) and especially when loops are nested inside other loops.

Having said this, it is a lamentable fact that most BASICS give very little option about how you lay out the program. In this respect, the compiled languages like PASCAL are far more flexible in that they are usually written with a text editor (or word processor). On the other hand, editing a BASIC program is generally a rather crude affair (unless, like Microsoft's MBASIC, your interpreter will take an ASCII version of the program and 'tokenise' it to turn it into a runnable program). Worse still, many BASICS will take the programs you write and reformat them to remove indentation! Some, on the other hand, will add indentation for you. The BBC Micro is quite good at this, but you have to remember to give it the LISTO command. Most PASCAL systems will include a formatter and they are generally very useful. However, for the sake of your own clarity of thought, it is a good idea to devise some formatting conventions, within the limits of your language.

Comments are, of course, the main way of documenting your programs within the programs themselves. Again conventions vary from

language to language. BASIC uses the REM statement. The word REM must appear at the front of your comment and then the interpreter will ignore everything it finds up to the next end-of-statement marker (: or (cr)). In other languages (PASCAL, PL/1, PROLOG, etc.) comments are bracketed by /* and */ (sometimes { and }), and anything between the marks is ignored by the compiler. An advantage of this system is that comments can run over more than one line. The disadvantage is that, if you forget the second */, the rest of your program is taken to be a comment and will be ignored!

Use comments wherever you feel some explanation may be needed: when you are defining constants, initialising variables, beginning a program, beginning a new procedure (subroutine), defining a function, or writing some code that isn't readily understood because of its complexity. Comments need not be long or wordy, and often just a reminder is needed. When you are trying to understand the logic of last year's adventure program, large blocks of general comment that break up the code and do not give enough detail can be more of a hindrance than a help, so keep comments short and to the point. Put them before tricky sections of code, and only put them inside the code when their presence is not likely to interfere with reading the logical structure of the program. Our final program (Listing 3) shows some examples as guidelines.

External documentation, in the form of handbooks and written specifications, is the hardest and most tedious to produce. For programmers, studies have shown that written documentation is usually only consulted as a last resort. However, when it is used, it can save a lot of effort. If your program is not too long and is well documented internally, it is unlikely that you will ever find a need for external *program* documentation. *User* documentation is another matter and will be discussed later in the series. Nonetheless, it is often useful to have some written documentation to hand when it comes to revising an old program or to debugging a new one. One of the ways in which the so-called 'fifth generation' languages aim to improve programmer productivity is by generating the documentation automatically. This will be achieved by using information from the design phase of a program's development. Not surprisingly, one of the best ways to document your own programs is to use this same trick.

Keep a file on your programs as you write them. Put into it all the notes you make as you design the program, including drafts of algorithms and flowcharts. Most importantly, keep the final version of the flowchart you have used to write the code from. If you have a printer, keep a listing of the finished program. Note that, in our completed version of the program, the first comment includes the program name and a date. Whenever you modify a program, change the date on it so that you know that it is the latest version.

Properly Documented

Listing 1

BASIC

```
(a) 10 INPUT A,B
    20 C=A*31536000
    30 D=B*2592000
    40 E=C+D
    50 PRINT E
```

PASCAL

```
(b) program abcde (input,output);
    var a,b,c,d,e:integer;
    begin
    read(a,b);
    c:=a*31536000;
    d:=b*2592000;
    e:=c+d;
    writeln(e);
    end.
```

Listing 2

BASIC

```
(a) 10 AYEAR=31536000
    20 AMONTH=2592000
    30 PRINT"Enter your age (years then months separated by a comma) ";
    40 INPUT NYEARS,NMONTHS
    50 YSECS=NYEARS*AYEAR
    60 MSECS=NMONTHS*AMONTH
    70 AGEINSECS=YSECS+MSECS
    80 PRINT"Your age in seconds is (approximately) ";AGEINSECS
```

PASCAL

```
(b) program ageinseconds (input,output);
    const
      ayear=31536000;
      amonth=2592000;
    var
      nyears,nmonths,ysecs,msecs,ageinsecs:integer;
    begin
      write('Enter your age (years then months separated by a comma) ');
      read(nyears,nmonths);
      ysecs:=nyears*ayear;
      msecs:=nmonths*amonth;
      ageinsecs:=ysecs+msecs;
      writeln('Your age in seconds is (approximately) ',ageinsecs);
    end.
```

Listing 3

BASIC

```
(a) 10 REM "AGEINSECONDS"   June 1984
    20 REM INPUTs age in years and months (y,m) and
    30 REM uses an approximate conversion (month = 30 days)
    40 REM to give age in seconds.
    50 REM
    60 AYEAR=31536000:REM         seconds in 365 days
    70 AMONTH=2592000:REM         seconds in 30 days
    80 PRINT"Enter your age (years then months separated by a comma) ";
    90 INPUT NYEARS,NMONTHS
   100 REM age in secs is (age in years * secs in year) plus
             (months since last birthday * secs in month)
   110 YSECS=NYEARS*AYEAR
   120 MSECS=NMONTHS*AMONTH
   130 AGEINSECS=YSECS+MSECS
   140 PRINT"Your age in seconds is (approximately) ";AGEINSECS
```

PASCAL

```
(b) program ageinseconds (input,output);
    /*   June 1984
         reads age in years and months (y,m) and uses an
         approximate conversion (month = 30 days) to give
         age in seconds.                                  */

    const
      ayear=31536000;      /* seconds in 365 days */
      amonth=2592000;      /* seconds in 30 days  */
    var
      nyears,nmonths,ysecs,msecs,ageinsecs:integer;
    begin
      write('Enter your age (years then months separated by a comma) ');
      read(nyears,nmonths);
      /* age in secs is (age in years * secs in year) plus
         (months since last birthday * secs in month)     */
      ysecs:=nyears*ayear;
      msecs:=nmonths*amonth;
      ageinsecs:=ysecs+msecs;
      writeln('Your age in seconds is (approximately) ',ageinsecs);
    end.
```

THE NUCLEAR FAMILY

A nuclear holocaust may hardly seem a suitable subject for a computer game, and many people may find the fantasy world of this game objectionable. But, ethical questions aside, Apocalypse — The Game of Nuclear Devastation is a traditional Diplomacy-style package that demands tactical skill and quick reactions.

Most computer games are solitary, anti-social diversions. Advertisements may depict families sitting grouped around the home computer, but games are generally designed for the single player.

However, Apocalypse brings back the traditional rivalry and intrigue associated with Diplomacy and other such board games, and thus is eminently suited to family gatherings. Indeed, the BBC version allows up to 15 players to take part. It is a genuine multi-player game, not simply one in which players take turns to participate in an essentially single-person game.

Apocalypse was originally a board game, based on Diplomacy, which has now been extended to take advantage of the graphics and number-crunching capabilities of the microcomputer. The main game covers four 'theatres of war' — Europe, the Caribbean, the UK and London — while extension kits may be purchased to include action in South Africa, the Levant, the Arctic, the USA, South-East Asia, the Pacific Campaign of World War Two, outer space, and historical campaigns based on the Fall of the Roman Empire and Napoleon's battles. These options are available on three tapes that are merged with the main game.

The screen is divided into a series of squares, displayed as a 40 by 20 (BBC) or 20 by 20 (Spectrum) matrix. Blue squares are used to represent sea, while various other colours depict different geographical features — rural or urban areas, cities, mountains or deserts. There are also symbols to represent the opposing forces. These graphics are dependent on machine capabilities. The Spectrum's simpler graphics have the benefit of clarity: the squares are either filled in or left empty, so it is easy to pick out territory that is occupied by an attacking force. The BBC display, while allowing greater graphic detail, appears somewhat cluttered by comparison and players' symbols tend to become lost in the background.

Players take turns to deploy their forces in a manner appropriate to the chosen scenario. Armies and navies are moved around the playing area, and attacks set up and repelled, with the computer acting as a referee. Moves are made by selecting options from various menus, and here the display could be improved to make things easier. The 'nuke' option may be selected in the main game, with predictably devastating results, but this may be used only in its correct historical context — the program won't allow you to launch a pre-emptive nuclear strike against unfriendly Roman legions, for example.

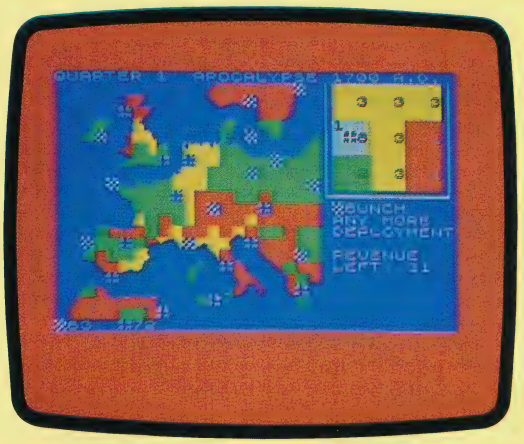
This is a lengthy game but, if enough like-minded players can be found, it is well worth investigating. Whether or not it is any better for being played on a microcomputer is another matter. Many people may find the board game equally compelling.

Apocalypse: for BBC Micro (2-15 players), £9.95
for ZX Spectrum (2-4 players), £9.95
Publishers: Red Shift, 12 Manor Road, London N16
Authors: Helmut Watson (BBC); Bob Tyler (Spectrum)
Original Game Design: Mike Hayes
Expansion Kits: £4.95 each
Joysticks: Not required
Format: Cassette

War Zones

Apocalypse is a game for all the family because up to 15 people can play at once on the BBC version, or up to four on the Spectrum version shown here. At least two players are always needed.

The players take up armed struggle that can eventually end in nuclear conflict. Thankfully, the display restricts itself to maps such as these, rather than showing pictures of devastation



LIZ HEANEY

DIRECTING THE ACTION

Good arcade games need to be written, at least in part, in machine code. This is a challenge for the beginner, so here we present a machine code sprite routine for the Spectrum. It can be used in two ways — either incorporated in a BASIC program by those who don't understand machine code, or used as a starting point by those who do.

Spectrum BASIC has many limitations, and these are especially noticeable when moving graphics are needed. Action games require the use of sprites of various shapes and sizes; these should be capable of smooth movement in all directions.

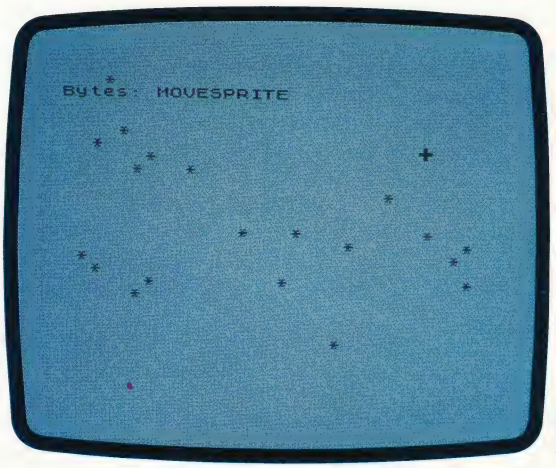
It is not easy to write graphics routines in Assembly language, but the program presented here should give you some good ideas on how to approach the task. The program prints a background of randomly placed asterisks, and allows you to move a user-defined shape (our example uses a cross) around the screen by pressing the unshifted cursor keys. The cross moves in smooth one-pixel steps, up, down, right or left, and leaves the background unchanged. The sprite-moving routine may easily be incorporated into your BASIC programs.

To instal the program in your Spectrum, you should first type in the BASIC program. The machine code may then be entered, either by typing in the BASIC loader program, which reads the code from data statements and then POKEs it into memory, or by entering the Assembly language source code by way of a suitable assembler. Both BASIC and machine code may be saved on cassette with lines 9000 and 9010 of the BASIC program.

To understand how the program works we will start by looking at the BASIC program. The subroutine at line 1000 reads the definition of the sprite from the data statements and POKEs this into RAM where it can be used by the machine code program. Lines 90 to 110 print the background, and line 120 sets the starting position for the cross. PRINT AT 10,16 makes the BASIC interpreter calculate the screen address corresponding to these character co-ordinates, and this address is stored in the system variable DFCC (addresses 23684 and 23685) where it can be read by the machine code program. Line 130 calls the initialisation section of the machine code program. Lines 140 to 180 are a loop that waits for a key to be pressed, POKEs the key value into a memory location for the machine code to read, and then calls the machine code to move the sprite one pixel in the direction indicated by the key.

The Assembly language program begins by defining names for the memory locations used. KEY is where the key value is stored. SPRPOS is used to hold the memory address of the screen position at which the sprite will appear. SPRTAB is a table in which the program stores the definition of the sprite and the contents of the screen locations that have been overwritten by the sprite. The sprite can be moved anywhere on the screen, not merely in jumps of whole character squares. So the eight bits in each row of the sprite may be divided between two bytes of screen memory, and the table uses two bytes to store the eight bits, split in the same way as on the screen. The memory location BITPOS is used to store the number of bits that the sprite data has been moved from the start of the byte.

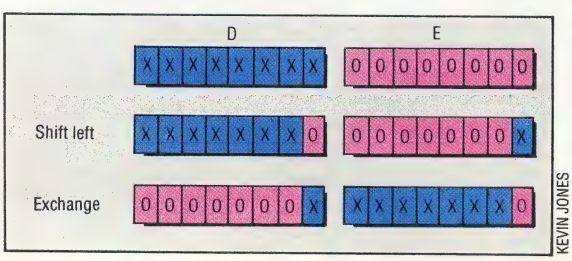
The initialisation section of the program reads the starting screen address of the sprite from DFCC, then jumps to the section labelled SAVSCR, where it stores the screen address in SPRPOS, loads the value 1 into the D register, and calls the subroutine UNDER. When D is set to 1, UNDER copies the contents of the screen area in which the sprite will be printed so that the background may be restored once the sprite has moved on. The program then calls the subroutine PRSPRT to print the sprite on the screen.



LIZ HEANEY

Sprite In Motion
The BASIC demonstration program moves the sprite (initialised as a cross in the DATA statements) across a background of stars in response to the unshifted cursor keys

The section of the program that handles the sprite movement starts at the label MOVSPR. It begins by loading 0 into the D register and calling



KEVIN JONES

Shift And Exchange
Shifting each of the sprite bytes (represented here by the Xs), in the DE register to effect left or right screen movement may cause a bit to 'wrap around'. If this happens, D and E are exchanged, re-uniting the sprite bits

Screen Addressing

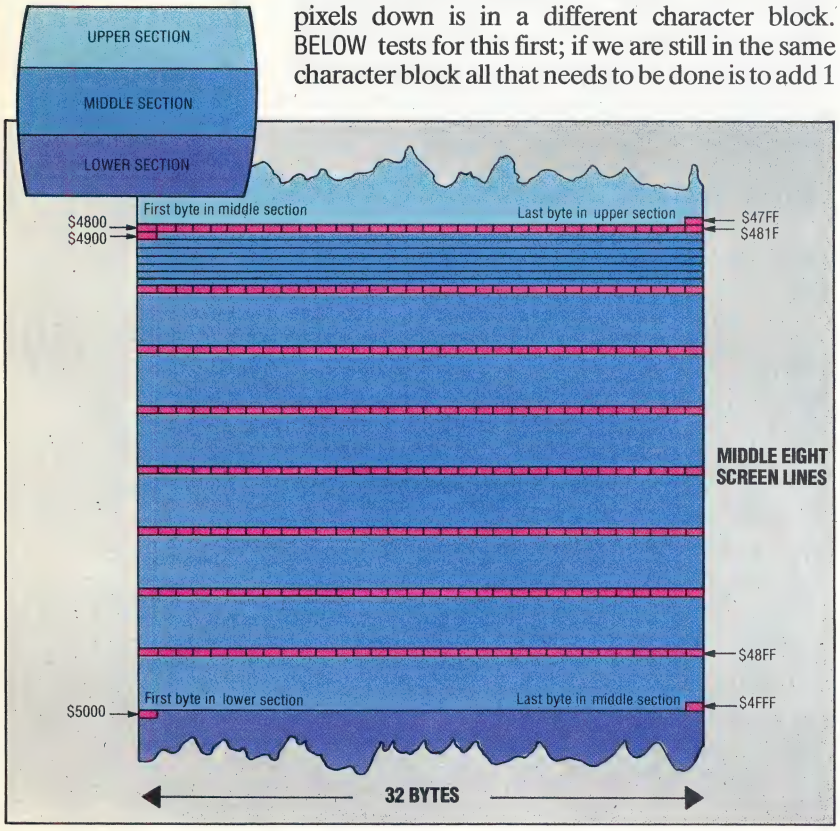
For memory-mapping purposes the Spectrum's 24-line screen is divided into three sections of eight screen lines; the details of the addressing of the middle section are shown here. Each line of the screen is divided into eight hi-res rows of 32 bytes, each bit of which corresponds to one dot pixel. Bytes are numbered sequentially along a hi-res row, and from each row in a screen line to the corresponding row in the screen line below: thus, the bytes in the eight top rows of the eight lines of one screen section have consecutive addresses. The next address is that of the first byte in the second row of the first screen line; all the bytes in the eight second hi-res rows in this screen section are addressed sequentially from this address, and so on. The address of the first byte in the top row of the first screen line of one section is one plus the address of the last byte in the eighth row of the eighth line of the preceding section. The following program for the 48K Spectrum illustrates this by POKEing a line into each byte of the hi-res screen in turn:

```
50 LET SCRNSTART=16384
60 LET SCRNEND=22527
100 FOR B=SCRNSTART TO
    SCRNEND
200 POKE B,255
300 NEXT B
```

the subroutine UNDER. With D set to 0, UNDER copies the previously saved screen contents back to the screen from the table, wiping out the sprite on the screen. The program then reads the sprite position and key value, tests the key value, calls the routine to prepare for movement in the appropriate direction and then jumps to SAVSCR to save the screen contents and print the sprite on the screen, just as before.

To understand the two routines ABOVE and BELOW that prepare for the sprite's up and down movement, we must first look at the odd way in which the Spectrum matches memory addresses to screen locations. This is explained in chapter 24 of the Spectrum manual. If you look at the addresses in hexadecimal you will see that for the 256 characters in each section of the screen the low byte of the address of each of the eight bytes that make up each character is the same as the character number within the block, while the high byte of the address increases by one when we move one line of pixels down the screen. For this reason, the eight rows of pixels for one character have addresses in the range $4000 to $47FF in the top third of the screen, from $4800 to $4FFF in the middle third of the screen, and from $5000 to $57FF in the bottom third of the screen. (The '$' sign means that numbers are in hexadecimal notation — some assemblers require the use of a hash (#) symbol instead.)

The subroutine BELOW expects a screen address in the HL register pair, calculates the address of the byte that is immediately below this position on the screen, and leaves this new value in HL. If you look at the screen addresses in binary you will see that when the three low bits of H are 111 the next row of pixels down is in a different character block. BELOW tests for this first; if we are still in the same character block all that needs to be done is to add 1 to H. If we are in a different character block we have to add $20 (since there are 32 characters on a line) to L. If the new value of L is between 0 and $1F (the three high bits 000) this means we are in a different screen section. The value of HL is the current screen address.



If we are still in the same screen block we have to subtract 7 from H. You will find this becomes easier to understand if you work through the code and see what it does to the addresses shown in the table. ABOVE is similar to BELOW, but calculates the address of the pixel above the screen position.

The subroutines LMOVE and RMOVE shift the pixel bit pattern in the table left and right. Again, they are very similar, so we will just look at how LMOVE works. The accumulator is loaded with the bit position pointer, which is a single-byte number in the range 0 to 7, corresponding to the numbering of bits in a byte. 1 is then subtracted from the accumulator value to effect the move; this will also result in a number in the range 0 to 7 unless the original value of the bit position pointer was 0, in which case the accumulator will hold 255. Use of the AND 7 instruction will now ensure that the value in the accumulator remains in the 0 to 7 range. We then have a loop for the eight rows of pixels in the sprite. For each row, we load the two bytes of the table containing that row of pixels into the DE register pair and perform a 16-bit rotate left on DE. If this does not move a bit of the sprite off the top end of D into the bottom end of E we can simply store the shifted sprite pattern back into the table and go on to the next row of pixels. If we have moved a bit of the sprite from the top of D to the bottom of E we need to exchange D and E before storing them back in the table. At the end of the routine we must also subtract 1 from HL so that the sprite will be printed one position to the left.

The final subroutine in the program is PRSPRT, which does the actual printing of the sprite on the screen. This consists of two nested loops, one for the eight rows of pixels in the sprite, controlled by the C register, and one for the two bytes the row of pixels is split between, controlled by the B register. The important part of the routine is the central section that stores the bits of the sprite pattern on the screen without disturbing those bits already on the screen that should not be covered by the sprite. We have the screen address in the HL register pair and the sprite table address in the IX register. PRSPRT takes a byte of the pixel pattern of the sprite, and ORs it with what is already on the screen so that we end up with the black dots from the sprite pixel pattern superimposed on the previous screen contents.

This program is not comprehensive — in particular the maximum size of a sprite is limited to eight by eight pixels, only one sprite is allowed, and the sprite does not carry its own colours around with it. However, if you understand how this program works you will be able to extend it to include extra features. Even without modification, it is a very useful addition to many BASIC and machine code programs.

KEVIN JONES

Sprite Moves

```
  5 REM   *Program 1*
 10 CLEAR 45055: REM AFFF HEX
 20 LOAD "MOVESPRITE"CODE
 30 LET KEY=45056
 40 LET BITPOS=45059
 50 LET SPRTAB=45060
 60 LET INIT=45312: REM B100 HEX
 70 LET MOVSPR=45317: REM B105 HEX
 80 GO SUB 1000: REM SET UP SPRITE
 90 FOR I=1 TO 20
100 PRINT AT 21*RND,31*RND;"*";
110 NEXT I
120 PRINT AT 10,16;
130 RANDOMIZE USR INIT
140 LET X$=INKEY$: IF X$="" THEN  GO TO 140
150 LET X=VAL X$
160 POKE KEY,X
170 RANDOMIZE USR MOVSPR
180 GO TO 140
1000 POKE BITPOS,0
1010 FOR I=0 TO 7
1020 READ X
1030 POKE SPRTAB+2*I,X
1040 POKE SPRTAB+1+2*I,0
1050 NEXT I
1060 RETURN
2000 DATA BIN 00011000
2010 DATA BIN 00011000
2020 DATA BIN 00011000
2030 DATA BIN 11111111
2040 DATA BIN 11111111
2050 DATA BIN 00011000
2060 DATA BIN 00011000
2070 DATA BIN 00011000
```

```
  5 REM   *Program 2*
 10 LET A=45312
 20 FOR L=1000 TO 1420 STEP 10
 30 LET S=0
 40 FOR A=A TO A+7
 50 READ B
 60 POKE A,B
 70 LET S=S+B
 80 NEXT A
 90 READ C
100 IF C<>S THEN  PRINT "ERROR IN LINE ";L: STOP
110 NEXT L
120 READ B
130 POKE A,B
140 READ B
150 POKE (A+1),B
200 PRINT "INSERT PROGRAM TAPE"
250 SAVE "MOVESPRITE"CODE 45312,400
1000 DATA 42,132,92,24,44,22,0,205,561
1010 DATA 61,177,42,1,176,58,0,176,691
1020 DATA 254,5,32,5,205,165,177,24,867
1030 DATA 24,254,6,32,5,205,109,177,812
1040 DATA 24,15,254,7,32,5,205,136,678
1050 DATA 177,24,6,254,8,192,205,228,1094
1060 DATA 177,34,1,176,22,1,205,61,677
1070 DATA 177,205,35,178,201,42,1,176,1015
1080 DATA 221,33,4,176,14,8,229,6,691
1090 DATA 2,203,66,32,6,221,126,16,672
1100 DATA 119,24,4,126,221,119,16,35,664
1110 DATA 205,83,178,221,35,16,234,225,1197
1120 DATA 13,200,221,35,197,213,205,109,1193
1130 DATA 177,209,193,24,217,62,7,164,1053
1140 DATA 254,7,40,2,36,201,17,32,589
1150 DATA 0,25,62,224,165,32,4,205,717
1160 DATA 83,178,201,124,214,7,103,201,1111
1170 DATA 62,7,164,40,2,37,201,17,530
1180 DATA 32,0,167,237,82,62,224,165,969
1190 DATA 254,224,32,4,205,76,178,201,1174
1200 DATA 124,198,7,103,201,221,33,3,890
1210 DATA 176,221,126,0,61,230,7,221,1042
1220 DATA 119,0,79,221,35,6,8,221,689
1230 DATA 94,0,221,86,1,203,3,245,853
1240 DATA 203,11,241,203,18,203,19,62,960
1250 DATA 7,185,32,3,122,83,95,221,748
1260 DATA 115,0,221,114,1,221,35,221,928
1270 DATA 35,16,220,62,7,185,192,43,760
1280 DATA 205,76,178,201,221,33,3,176,1093
1290 DATA 221,126,0,60,230,7,221,119,984
1300 DATA 0,79,221,35,6,8,221,94,664
1310 DATA 0,221,86,1,203,11,245,203,970
1320 DATA 3,241,203,26,203,27,62,0,765
1330 DATA 185,32,3,122,83,95,221,115,856
1340 DATA 0,221,114,1,221,35,221,35,848
1350 DATA 16,220,62,0,185,192,35,205,915
1360 DATA 83,178,201,42,1,176,221,33,935
1370 DATA 4,176,14,8,229,6,2,221,660
1380 DATA 126,0,47,87,126,162,221,182,951
1390 DATA 0,119,35,205,76,178,221,35,869
1400 DATA 16,237,225,13,200,197,205,109,1202
1410 DATA 177,193,24,224,62,63,188,192,1123
1420 DATA 38,87,201,62,88,188,192,38,894
1430 DATA 64,201,265
```

```
KEY     EQU  £B000
SPRPOS  EQU  £B001
BITPOS  EQU  £B003
SPRTAB  EQU  £B004
DFCC    EQU  £5C84
        ORG  £B100
INIT    LD   HL,(DFCC)
        JR   SAVSCR
MOVSPR  LD   D,0
        CALL UNDER
        LD   HL,(SPRPOS)
        LD   A,(KEY)
        CP   5
        JR   NZ,L0
        CALL LMOVE
        JR   SAVSCR
L0      CP   6
        JR   NZ,L1
        CALL BELOW
        JR   SAVSCR
L1      CP   7
        JR   NZ,L2
        CALL ABOVE
        JR   SAVSCR
L2      CP   8
        RET  NZ
        CALL RMOVE
SAVSCR  LD   (SPRPOS),HL
        LD   D,1
        CALL UNDER
        CALL PRSPRT
        RET
UNDER   LD   HL,(SPRPOS)
        LD   IX,SPRTAB
        LD   C,8
LINE    LD   B,2
BYTE    BIT  0,D
        JR   NZ,SVESCR
WIPOUT  LD   A,(IX+£10)
        LD   (HL),A
        JR   CONT
SVESCR  LD   A,(HL)
        LD   (IX+£10),A
CONT    INC  HL
        INC  IX
        DJNZ BYTE
        DEC  C
        RET  Z
        INC  IX
        DEC  HL
        DEC  HL
        PUSH BC
        PUSH DE
        CALL BELOW
        POP  DE
        POP  BC
        JR   LINE
BELOW   LD   A,7
        AND  H
        CP   7
        JR   Z,BDIFCB
BSAMCB  INC  H
        RET
BDIFCB  LD   DE,£20
        ADD  HL,DE
        LD   A,£E0
        AND  L
        RET  Z
BSAMSB  LD   A,H
        SUB  7
        LD   H,A
        RET
ABOVE   LD   A,7
        AND  H
        JR   Z,ADIFCB
ASAMCB  DEC  H
        RET
ADIFCB  LD   DE,£20
        AND  A
        SBC  HL,DE
        LD   A,£E0
        AND  L
        CP   £E0
        RET  Z
ASAMSB  LD   A,H
        ADD  A,7
        LD   H,A
        RET
LMOVE   LD   IX,BITPOS
        LD   A,(IX+0)
        DEC  A
        AND  7
        LD   (IX+0),A
        LD   C,A
        INC  IX
RMOVE   LD   IX,BITPOS
        LD   A,(IX+0)
        INC  A
        AND  7
        LD   (IX+0),A
        LD   C,A
        INC  IX
        LD   B,8
RPAIR   LD   E,(IX+0)
        LD   D,(IX+1)
        RRC  E
        PUSH AF
        RLC  E
        POP  AF
        RR   D
        RR   E
        LD   A,0
        CP   C
        JR   NZ,RSTORE
        LD   A,D
        LD   D,E
        LD   E,A
RSTORE  LD   (IX+0),E
        LD   (IX+1),D
        INC  IX
        INC  IX
        DJNZ RPAIR
        LD   A,0
        CP   C
        RET  NZ
        INC  HL
        RET
PRSPRT  LD   HL,(SPRPOS)
        LD   IX,SPRTAB
        LD   C,8
PRLINE  LD   B,2
PRBYTE  LD   A,(IX+0)
        CPL
        LD   D,A
        LD   A,(HL)
        AND  D
        OR   (IX+0)
        LD   (HL),A
        INC  HL
        INC  IX
        DJNZ PRBYTE
        DEC  C
        RET  Z
        DEC  HL
        DEC  HL
        PUSH BC
        CALL BELOW
        POP  BC
        JR   PRLINE
        LD   B,8
LPAIR   LD   E,(IX+0)
        LD   D,(IX+1)
        RLC  E
        PUSH AF
        RRC  E
        POP  AF
        RL   D
        RL   E
        LD   A,7
        CP   C
        JR   NZ,LSTORE
        LD   A,D
        LD   D,E
        LD   E,A
LSTORE  LD   (IX+0),E
        LD   (IX+1),D
        INC  IX
        INC  IX
        DJNZ LPAIR
        LD   A,7
        CP   C
        RET  NZ
        DEC  HL
        RET
```

How To Use These Programs

1) Type in and SAVE "Program 1" LINE 10
2) Type in and RUN *Program 2*: this SAVEs the machine code from memory, preferably on the tape after *Program 1*
3) LOAD "Program 1", which will auto-run

RADIO CONTACT

Motorola is a company that is most often associated with car radios. But, from humble beginnings, Motorola Incorporated has grown to its present position as one of the world's leading microelectronic component manufacturers, with factories in Europe and the United States producing microprocessors for the 16-bit market.

Robert Galvin, Motorola's Chairman

Like many other successful business concerns, Motorola began as a one-man business. The company dates back to 1928, when Paul Galvin founded the Galvin Manufacturing Corporation in Chicago, where he specialised in the production of domestic radios. During the 1930s the company diversified, manufacturing police and car radios under the brand name 'Motorola'. In the 1940s the corporation — now known as Motorola Incorporated — was one of the first electronics firms to produce semiconductors.

Paul Galvin died in 1959, and was succeeded as chairman by his son, Robert. During the ensuing decade other manufacturers, notably those in Japan, began to compete with Motorola in the semiconductor and consumer electronics markets. The world recession of the mid-1970s led to enormous losses for the company, and Motorola was forced to rethink its strategy. New personnel were hired, many coming from Motorola's arch-rival, Texas Instruments, and the decision was taken to abandon the more traditional electronics field — in which the company could no longer compete — and instead concentrate on high technology microelectronics.

This involved the sale of some company assets — notably the colour television business — the investment of large sums in research and development, and the purchase of companies in new areas where Motorola hoped to make an impact. This was a considerable risk, but then the company had little choice in the matter.

The gamble seems to have paid off. During the latter part of the 1970s Motorola lagged far behind the leading firms in the semiconductor market, but after heavy investment in new technology the company can now claim to be breathing down the neck of market leader Texas Instruments. As Robert Galvin says: 'Companies that used to be competitors to Motorola aren't around any more, because they haven't adapted to the environment.'

Motorola has continued to have problems in making its products available at the right time. In the mid-1970s, when the microcomputer industry was in its infancy, the Motorola 6800 eight-bit microprocessor was outsold by the Mostek 6502 chip, which was adopted by Apple for its hugely successful personal computers, and by the Intel 8085 and Zilog Z80 used on CP/M computers. The company introduced the 6809 in 1976; this was generally acknowledged to be the best available eight-bit microprocessor, but the race for the mass market had already been lost and the chip appeared in only a few home micros such as the Tandy Color Computer and the Dragon.

However, the company continued to invest heavily in research — 'to get maximum advantage soonest', according to Robert Galvin — and is far better placed in the race for the 16-bit market. The 68000 microprocessor was launched in 1979, although it did not become widely available until 1982. This processor has been adopted for Apple's Lisa and Macintosh microcomputers and by Sinclair Research for the QL. It is an extremely powerful device containing 17 32-bit registers, a 16-bit data bus and a 24-bit address bus.

Motorola's Headquarters, Illinois, USA

Motorola continues to develop new products from its research centres in Phoenix, Arizona, Geneva in Switzerland and East Kilbride, Scotland. The East Kilbride factory produces CMOS (Complementary Metal Oxide Semiconductors) and MOS (Metal Oxide Semiconductors) chips for a wide range of applications. The company is now organised into five groups, dealing with communications, semiconductors, information systems, automotive and industrial electronics and government electronics. Despite some concern over low profitability in some sectors, the company reported sales of $1.26 billion in the first quarter of 1983 and seems set to maintain its strong position in the market.

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

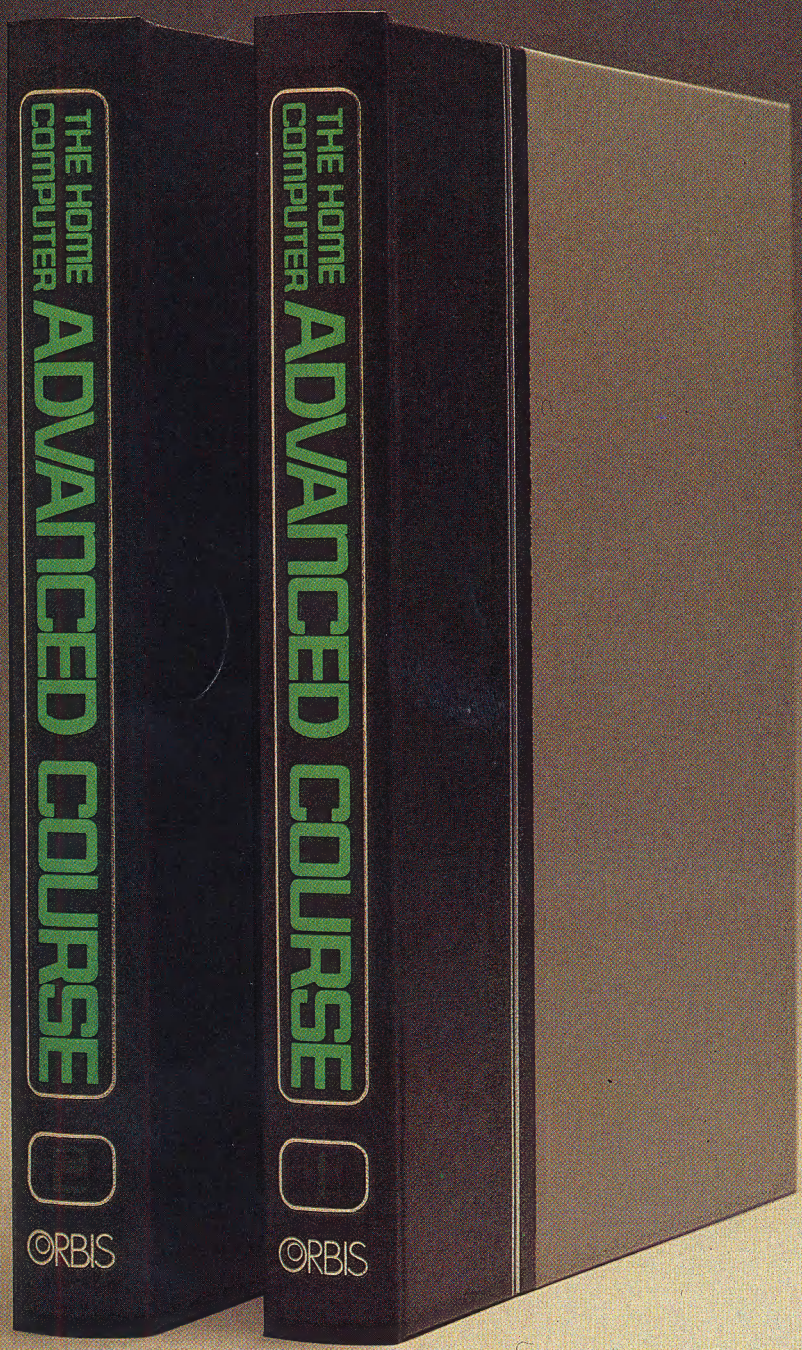
No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

THE HOME COMPUTER ADVANCED COURSE

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.



All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.

Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.

The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

PLACE A STANDING ORDER TODAY.